MASTER IN COMPUTER SCIENCE

QROSS-checking RESCUE models

Moyen, Catherine

Award date:
2005

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Année académique 2004-2005

QROSS-Checking RESCUE models

Moyen Catherine

Mémoire présenté en vue de l'obtention du grade de Maître en Informatique

Abstract

This thesis takes place in the field of the Requirements Engineering. This area of Information Systems Engineering is dedicated to discover the needs that users and other stakeholders have towards a system under (re)design. To support this activity, City University in London has developed a process called RESCUE, which stands for Requirements Engineering with Scenarios for User-Centered Engineering. Our work consists in automating some synchronisation checks between models and/or templates taking place at defined moments in this process, by extending a tool developed by the CETIC (Centre of Excellence in Information and Communication Technologies), QROSS.

Ce mémoire relève du domaine de l'Ingénierie des Exigences. Cette branche de l'Ingénierie des Systèmes d'Information a pour but de découvrir les besoins des utilisateurs et autres acteurs d'un système en phase de (re)conception. Il aborde en particulier la méthode RESCUE (Requirements Engineering with Scenarios for User-Centered Engineering) développée à la City University, à Londres, qui supporte les activités de l'ingénierie des exigences. Cette méthode est structurée en phases. Nous développons ici l'automatisation des contrôles de synchronisation qui ont lieu à la fin de chacune des phases de la méthode, et qui comparent différents modèles ou grilles (templates) entre eux. Pour cela, nous utilisons un outil appelé QROSS, développé par le CETIC (Centre d'Excellence en Technologies de l'Information et de la Communication).

Acknowledgements

We would like to thank all persons who provided their assistance for the realization of this thesis:

Prof Patrick Heymans from University of Namur, for his support and guidelines;

Prof Neil Maiden, Dr Sara Jones and Dr Cornelius Ncube at City University, London, for their support, their warm welcome and their patience;

Gaëtan Delannay at CETIC for his support and advices;

Julien Stouffs, Lionel Deliege and Thomas de Bodt for their support and their help for the second reading;

My family, my friends and all the people who were there when i needed to be cheered up.

QROSS-checking RESCUE models

Abstract

This thesis takes place in the field of the Requirements Engineering. This area of Information Systems Engineering is dedicated to discover discover the needs that users and other stakeholders have towards a system under (re)design. To support this activity, City University in London has developed a process called RESCUE, which stands for Requirements Engineering with Scenarios for User-Centered Engineering. Our work consists in automating some synchronisation checks between models and/or templates taking place at defined moments in this process, by extending a tool developed by the CETIC (Centre of Excellence in Information and Communication Technologies), QROSS.
The present thesis is structured as follow: chapter one is dedicated to introducing basic notions of Requirement Engineering; in chapter two, we introduce the RESCUE process; chapter three presents a system already designed following the RESCUE process, used as a case study; chapter four presents the QROSS tool and the architecture of the extensions made to it; chapter five describes the implementation of the checks on top of the existing QROSS infrastructure; chapter six presents the results of the automation on the case study introduced in chapter two; and finally, chapter seven is dedicated to evaluate the work we did and how we envisage its future evolution.

Acknowledgements

We would like to thank all persons who provided their assistance for the realization of this thesis:

Prof Patrick Heymans from University of Namur, for his support and guidelines;

Prof Neil Maiden, Dr Sara Jones and Dr Cornelius Ncube at City University, London, for their support, their warm welcome and their patience;

Gaetan Delannay at CETIC for his support and his advices;

Julien Stouffs, Lionel Deliege and Thomas de Bodt for their support and their help for the second reading.

Table of Contents

Chapter 1

Introduction: Requirements Engineering

1.1 Short definition

Today, it's a well-know fact that Requirements Engineering (RE) is a crucial phase of the development of a computer-based system. RE is dedicated to find out *what* a computer based system should do, as opposed to *how* it should do it, which is the aim of the software engineering. As Boehm said in 1981 [Boeh 81], RE is "*designing the right thing*", as opposed to Software Engineering, which is "*designing the thing right*". Naturally, RE and SE are both needed to develop a useful system. Figure 1.1 shows how they are articulated in a simple waterfall life-cycle model.

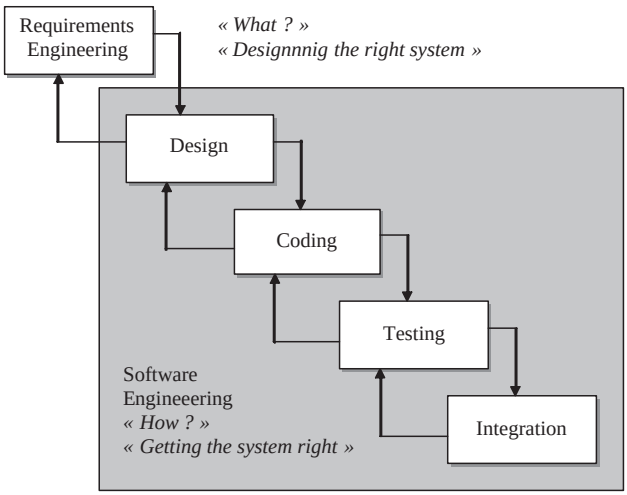

Figure 1.1: Standard waterfall life-cycle model

1.2 Why is RE important?

There are many reasons why RE is so important in the development of a computer system.
The main reason is described as follows in [Sutc 02]:

> *"Computers systems have made the requirement problem worse because we build systems in many different domains. The requirements engineer (or designer) has to understand not only what the user wants, but also the implications of the domain and what is achievable. Since gathering requirements inevitably involves communication between people, and natural language is prone to misinterpretation, requirements analysis has been a frequent cause of system of failure."*

The reduction of the costs generated by such errors constitutes a major motivation for RE.
Examples of system failures due to a wrong definition of user needs are numerous enough to prove the importance of RE. Among well known examples of such problems, we can cite the London Ambulance Service's Computer Aided Dispatch System. It had been designed to replace the manual system of answering emergency telephone calls and dispatching an available ambulance to the location of the emergency. Little requirements analysis was carried out with the users of the designed system. Several problems appeared with the use of the system: "radio blackspots where the ambulance crews could not be contacted, poor user interfaces on the mobile data terminals which resulted in the ambulance crews not reporting call progress accurately, with the knock-on effect that the system database became inaccurate, causing the automatic call location program to lose ambulances, dispatch ambulance crews who were not free, send several ambulances to the same call, and so on" [Sutc 02].
Another example of problems caused by bad requirements analyses is Eurocontrol (Europe-wide air traffic control). Requirements for flight co-ordination are complex and difficult to stabilize and finalize. The process that is the subject of this thesis has been used to find a solution to this problem of definition of the requirements for Eurocontrol.
RE provides strong basis to the next step in the system development, software engineering, which will design all that has been defined by RE.

1.3 Problems in RE

Requirements can be imposed by laws of physics and facts of nature, some are imposed externally as constraints on design required by law, and so on. But mostly, requirements come from people. This involves thus communication and its problems, such as tacit knowledge, the ambiguity of natural language, and other factors like the role of power, personality and opinions. The main issue of RE is so to transform informal and fuzzy statements of requirements into a formal specification that is understood and agreed by all stakeholders.

Another problem is that, theoretically, it should be possible to obtain nearly perfect requirements. However, including time and resource constraints of practice, this is unlikely. As the world keeps changing while defining requirements, these always refer to a passed state of the world and the system. RE helps here by producing documents which can have a contractual value: once this "contract" was accepted by the customer for whom the system is conceived, the developers commit themselves to provide a system that fulfills these requirements. Requirements are at best a compromise.

We conclude this section by citing [Sutc 02]:

> *"(...) at the heart of RE there is a paradox: users don't know what they want until they get it, and when they get it they see how it could be improved or they don't like it. Trying to overcome this 20/20 foresight problem lies at the heart of RE."*

1.4 Types of requirements

There are various types of requirements. Some may be high-level goals, others may be detailed rules and constraints. We give an example from [Sutc 02] to illustrate this:

> *"(...) the requirements in a library circulation system could include:*
>
> - *the need for a complete sub-system or high-level functions: "the system will have facilities for auditing book stock so that old and redundant stock can be eliminated";*
> - *specification of a more detailed function: "the circulation control system should calculates fines on overdue loans";*
> - *a statement about how a function should work: "fines should be calculated as the number of days overdue (current date - due date) multiplied by a configurable fine factor";*

- *constraints on how the system should operate: "data on reader fines should be secure and not publicly accessible";*
- *statements about performance: "all search requests must be completed within 30 seconds of submission";*
- *implementation constraints: "the system must operate on a Linux platform as well as Windows NT".*"

The major distinction is between functional and non-functional requirements. They are defined as follows in [Sutc 02]:

> "Functional requirements *are statements about what a system should do, how it should behave, what it should contain, or what components it should have. Functional requirements are initially expressed as goals, e.g. "the search facility will find books that match the user's request", or activities, e.g. "validate order", "monitor temperature". Later, requirements become specifications expressed as entities, attributes, actions, events or states (...). Functions are elaborated processes or procedures which can be implemented as software algorithms and data structure.* Non-functional requirements *(NFRs) are statements of quality, performance and environment issues with which the system must conform. Some examples are reliability, maintainability, portability (properties of the design), safety, security, scalability, accuracy, usability, and performance. NFRs are qualities and performance criteria that are not directly implementable in software. (...)*"

Functional requirements may be derived from non-functional ones. An example is given in [Sutc 02]: a designer having a non-functional requirement explaining that "the autopilot system will fly the aircraft to ensure no collisions occur" might translate what "no collisions" means as "the autopilot will detect any aircraft which comes within an envelope of 500 meters and take avoiding action".

1.5 Documents for requirements

As we already said, requirements are usually expressed in natural language, which is subject to misinterpretations.
A solution which was considered was the use of formal, mathematically based specification language. But such languages, although they make the meaning of the requirements less ambiguous, are not comprehensible to most users.
Semi-formal notation, such as i* models ([Yu 96]) or context models, are another solution, which can be coupled with more formal specification language to ease the comprehension of the latter.

A last solution considered is to structure requirements by the use of templates. A good example of requirements documentation template is given in the VOLERE method ([Robe 99], see table 1.1). Those templates can be used within requirements management tools like RequisitePro or DOORS.

Requirement #:	Requirement Type:	Event/use case #:
Description:		
Rationale:		
Source:		
Fit Criterion:		
Customer Satisfaction:	Customer Dissatisfaction:	
Dependencies:	Conflicts:	
Supporting Materials:		
History:		

Table 1.1: Requirements template from the VOLERE method.

1.6 Paper's structure

After having explained why RE is so important and some of its basis, we describe in the next chapters how the RESCUE process helps to define right requirements to provide a strong basis for implementation and how we automated some of its parts. Chapter 2 is dedicated to the explanation of how the RESCUE process works, giving its structure and an explanation about models and notation used in it. Chapter 3 gives the statement of the case study we used to evaluate the results of our work, including the models which had been developed by the team at City University. In chapter 4, we describe the tool we used to automate the phases of checks in the RESCUE process and develop the QROSS-Checker. We give the conceptual meta models for the various models used in the RESCUE process, and those we used to implement our tool. Chapter 5 describes precisely how we implemented the various checks. We sometimes give examples to make the concept clearer when needed. Chapter 6 gives the results of the application of the QROSS-Checker to the case study. Finally, chapter 7 concludes this thesis by evaluating the work we did and giving some tracks for the continuation of the work we began.

Chapter 2

The RESCUE process

2.1 Description

In this chapter, we will describe the RESCUE process. We initially point out that RESCUE is a process in constant evolution and that there are thus several versions of it. The version on which we worked is version 5 ([Maid 04b]).
To introduce the RESCUE process, we cite [Maid 04b]: *"The purpose of the RESCUE process, when applied in the context of a new project, is to guide the projects requirements engineering team to deliver a complete, correct and testable specification of requirements for a future system. It recognises real-world constraints on the process, and also supports the analysis of current work practices to inform the change that will arise from the introduction of the new system. In addition, it uses creative design processes to generate additional requirements and to underpin these requirements with high-level design alternatives."*
RESCUE has been conceived for the design of socio-technical systems that involve people as well as software. Thus, the process must design people's activities as well as the software systems that support peoples work. *"The RESCUE (...) process (...) supports a concurrent engineering process in which different modelling and analysis processes take place in parallel. The concurrent processes are structured into 4 streams (...)."*([Maid 04d])
Those vertical streams are shown in figure 2.1 from [Maid 04b]. These streams are described as follows in [Maid 04d]:

> *"Each stream has a unique and specific purpose in the specification of a socio-technical system:*
>
> 1. *Human activity modelling provides an understanding of how people work, in order to baseline possible changes to it ([Vice 99]);*
> 2. *System modelling enables the team to model the future system boundaries, actor dependencies and most important system goals ([Yu 94]);*

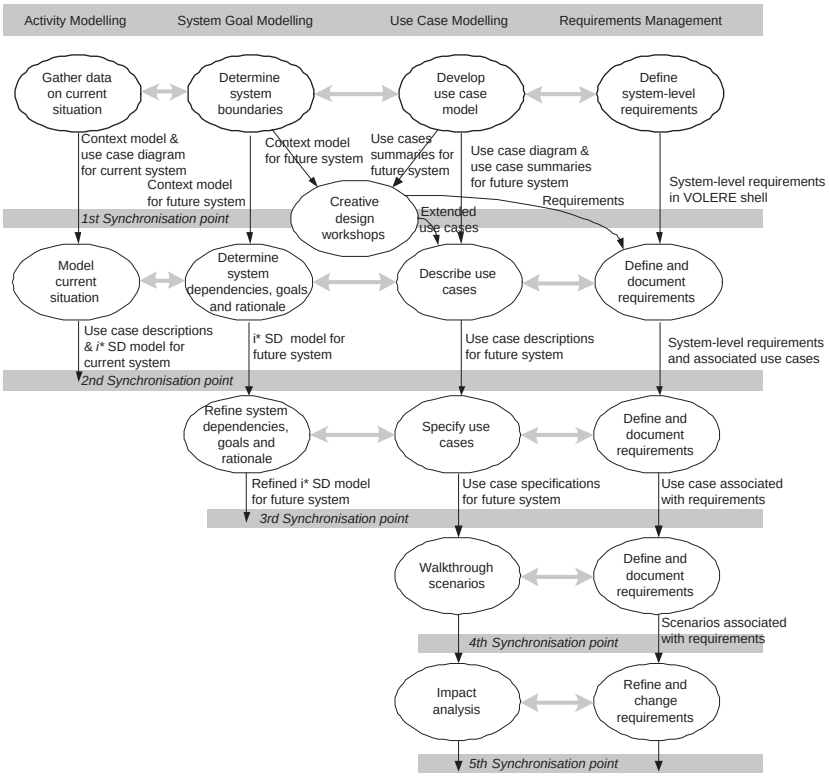

Figure 2.1: The RESCUE process structure

3. *Use case modelling and scenario-driven walkthroughs enable the team to communicate more effectively with stakeholders and acquire complete, precise and testable requirements from them ([Sutc 98]);*

4. *Managing requirements enables the team to handle the outcomes of the other 3 streams effectively as well as impose quality checks on all aspects of the requirements document ([Robe 99]);*

Sub-processes during these 4 streams are co-ordinated using 5 synchronisation stages that provide the project team with different perspectives with which to analyse system boundaries, goals and scenarios. These stages are implemented as synchronisation checks (. . .).”

These stages will be described further in this chapter, in section 2.3.

2.2 Models and documents

As we can see in figure 2.1, various models and documents are generated during the RESCUE process. For each model, we give an explanation about the notations, concepts and tools used. Additional guidelines to draw these models to make them suitable to the we have developed are given in appendix B.

2.2.1 Context Model

The first diagram produced is a context model. As said in [Maid 05a], it *"separates what the project team will design and what is beyond its scope (...)"* and *"provides the baseline for more complicated i* SD model and use case model"*. These latter models will be explained further in this chapter. The context model is presented as a data flow diagram, composed of actors (or agents[1]), divided into several levels:

1. the technical level contains the computer based systems to design or redesign;

2. the socio-technical level, which contains the primary users, i.e. the users whose work will be changed by the redesign of the technical system;

3. the "uncontrolled system" level contains systems and people that will change to accommodate the new system and its users, but not dependent to it;

4. the external level contains systems and people that do not change , which act independently of the work being studied but has connections to it.

The level 4 is not mandatory. So, it is possible to have context models showing only 3 levels.
These various levels are represented by rectangles included one in the other, the outermost corresponding to level 4 if any, at the level 3 otherwise, the innermost corresponding to level 1. Initially, these levels were represented using circles, but for the ease of computation of the position of actors within these levels in he phase of implementation of our tool, we chose rectangles to represent them.

Subsystems in the technical system (level 1) are represented by circles with the name of the subsystem in it. This is the original notation.

[1]The term "actor" is used interchangeably with the term "agent" and denotes people, systems or companies

Actors outside the technical system were initially represented as text statements. Once again, for the ease of computation, we chose to use rectangles with rounded corners with the name of the actor in it.

These actors interact. These interactions, taking place either between an actor and the new system, or between actors beyond the system boundary, are represented as data flows, by arrows labeled to describe the information that is flowing. The arrow head indicates the direction of the flow, the arrow head pointing to the actor or system receiving the information. The data flows can be one-way or two-way data flows.

Initially, RESCUE users built context models using MS Word. To make the implementation of parsers easier, we chose to use an already used tool in the RESCUE process: MS Visio (see section 4.5).

We can see an example of a context model on figure 3.1 on page 21. This example will be explained with more details in chapter 3

2.2.2 Use case model

A use case model consists of a use case diagram and a set of use case descriptions made according to a template. The two following subsections explain those concepts.

Use case diagram

This model is the well-known use case diagram defined in the UML ([Rumb 99]). So we will not explain further what a use case diagram is.
An example of a use case diagram can be found in figure 3.2, on page 22, with an explanation.

Use case template

A description should be produced for each use case in the use case diagram. Each description is done using the use case template defined in the table 2.1. This template was initially developed in the CREWS-SAVRE project ([Sutc 98]) and has been slightly modified for use in the RESCUE process. An example of a use case description using this template is provided by table 3.2 on page 26.

Use case ID	Unique ID for the use case
Name of use case	Name of the use case
Text	A text field describing the use case
Author	Name of the author
Date	Date use case was written
Source	Source of the use case
Actors	Actors involved in the use case
Problem statement (now)	Description of current problem
Precis	Informal scenario description
Functional requirements	Descriptions of all functional requirements associated with this use case
Non-functional requirements	Descriptions of all non-functional requirements associated with this use case
Added value	Benefit of Use Case above and beyond the original scenario from the original system
Justification	Why is the use case needed in the future system?
Triggering event	Event or events that can trigger the use case
Preconditions	Necessary conditions for the use case to occur
Assumptions	Explicit statement of any assumptions made in writing the use case
Successful end states	Successful outcome(s) of the Use Case
Unsuccessful end states	Unsuccessful outcome(s) of the Use Case
Different walkthrough contexts	Different situations and contexts in which the use case takes place
Normal course	1. Action 1 Descriptions of all functional and non-functional requirements related to Action 1
	2. Action 2 Descriptions of all functional and non-functional requirements related to Action 2
	...
Variations	1. If [condition] then [variation statement, identifying 1 or more actions in the normal course which are different in the condition given] Descriptions of all functional and non-functional requirements related to this Variation. List of one or more actions to replace relevant actions in the normal course, together with their associated functional and non-functional requirements.
	...
Satisfaction arguments	Maintained by RequisitePro
Option	Maintained by RequisitePro
Features	Maintained by RequisitePro

Table 2.1: Use case template

2.2.3 Strategic Dependency model (SD model)

This model describes the network of dependency relationships between actors. It is made using the i* goal modelling approach developed by Eric Yu in his PhD thesis ([Yu 96]). To support i* modelling, a plug-in to MS Visio called REDEPEND has been developed at City University. We describe here the notations used within this type of model:

- An actor is simply represented by a circle containing its name. The actor situated at the left side of the "D" we see on a dependency is called the *depender*, the actor who "wants" something. The actor on the right of the "D" is called the *dependee*, the actor who has the "ability" to provide something.

- The authors of [Maid 04b] define a goal as "*a condition or state of the world that can be achieved or not*". A goal dependency between two actors is represented as shown on figure 2.2. This must be interpreted as "the customer depends on the airline to have tickets bought". Here, the customer is the depender and the airline is the dependee.

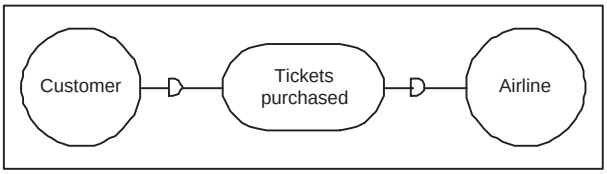

Figure 2.2: i* Goal Dependency

- The authors of [Maid 04b] explain that a soft-goal is "*used into i* to denote non functional requirements, relating to properties of the future system*". A soft goal dependency is shown on figure 2.3. Here, the student (depender) depends on the teacher (dependee) to learn well.

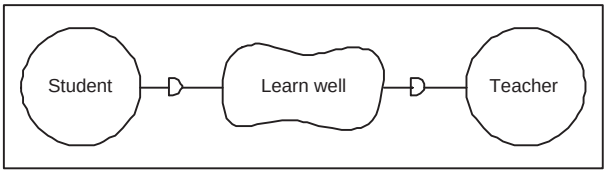

Figure 2.3: i* Soft Goal Dependency

- The following definition of a resource is given in [Maid 04b]: "*Means, including objects or tool available to actors to achieve goals in the problem domain. In i*, resources may be either physical, informational or time resources*". Figure 2.4 illustrates a resource dependency. The example given on the schema must be interpreted as "the passenger depends on the Countdown Indicator to have the bus arrival information".

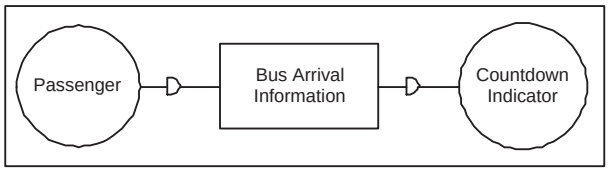

Figure 2.4: i* Resource Dependency

- "*In i* modelling, a task is a particular way of achieving a goal; a sequence of actions which produces a change in the problem domain*" ([Maid 04b]). Figure 2.5 gives an example of a task dependency. The interpretation here is: "the teacher depends on the student to deliver lecture".

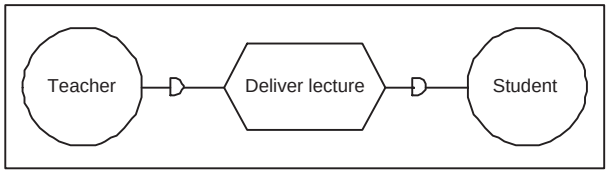

Figure 2.5: i* Task Dependency

An example of an SD model is given on figure 3.5 on page 24 and will be explained further.

2.2.4 Requirements template

A requirement is defined as follows in [Robe 99]: "Something that a product must do or a quality that the product must have". The requirements are divided in three different levels defined as follows in [Maid 04b]: system level requirements, "which relate to the system as a whole", use case level requirements, "which relate to a particular use case as a whole", and action level requirements, "which relate to an individual action in a use case".
The requirements template has been adapted from the VOLERE requirement template (see table 1.1) of J. and S. Robertson [Robe 99]. All the

requirements related to existing use cases should be described in a requirement template. In the RESCUE process, those requirement templates are fulfilled using RequisitePro. Table 2.2 presents this template[2].

Requirement tag	Unique ID
Requirement type	One of the allowable RESCUE requirements types
Use case	Relevant use case (if requirement comes from a use case)
Description	A one sentence statement of the intention of the requirement
Rationale	Why the requirement is considered important or necessary
Status	Proposed or Pending or Approved
Owner	Person responsible for the requirement
Source	Origin of the requirement
Stability	High or Medium or Low
Fit criterion	A quantification of the requirement used to determine whether the solution meets the requirement
Customer satisfaction	Measures the desire to have the requirement implemented on a scale 1 – 5
Customer dissatisfaction	Measures unhappiness if the requirement is not implemented on a scale 1 – 5
Dependencies	Other requirements with a change effect
Conflicts	Other requirements that contradict this one
Supporting materials	References to any explanatory information
History	Date requirement was raised, dates and details of changes

Table 2.2: Requirements template

2.3 Stages and synchronisation checks

As we said at the beginning of this chapter, the RESCUE process is divided into 5 stages, ended by a synchronisation point. These synchronisation points are implemented as checks which have to be satisfied, to insure that the bases for the following stage are right and to have the agreement of all stakeholders taking part in the development of the system. At these synchronisation points, the models produced during the completed stage are

[2]The various types of requirements used in the RESCUE process are given in appendix C

submitted to checks pairwise. These checks are done "manually": the RESCUE user analyses the relevant models and compares the artefacts related to the check. Figure 2.6 from [Maid 04c] presents the stages of the RESCUE process, and illustrates the checks as black arrows between diagrams or templates. The various statements of the checks we implemented are from [Maid 04c]. They do not exactly correspond to the checks in version 5 ([Maid 04b]) of the process. This choice was made because our work is based on a case study (see chapter 3 and [Maid 04c]) and we did not have the results for the checks of version 5. However, the checks are very similar in both versions, as only one check in the second stage has been added to version 5.

In the next subsections, we describe the stages and give some typical examples of checks done at the end of each stage.

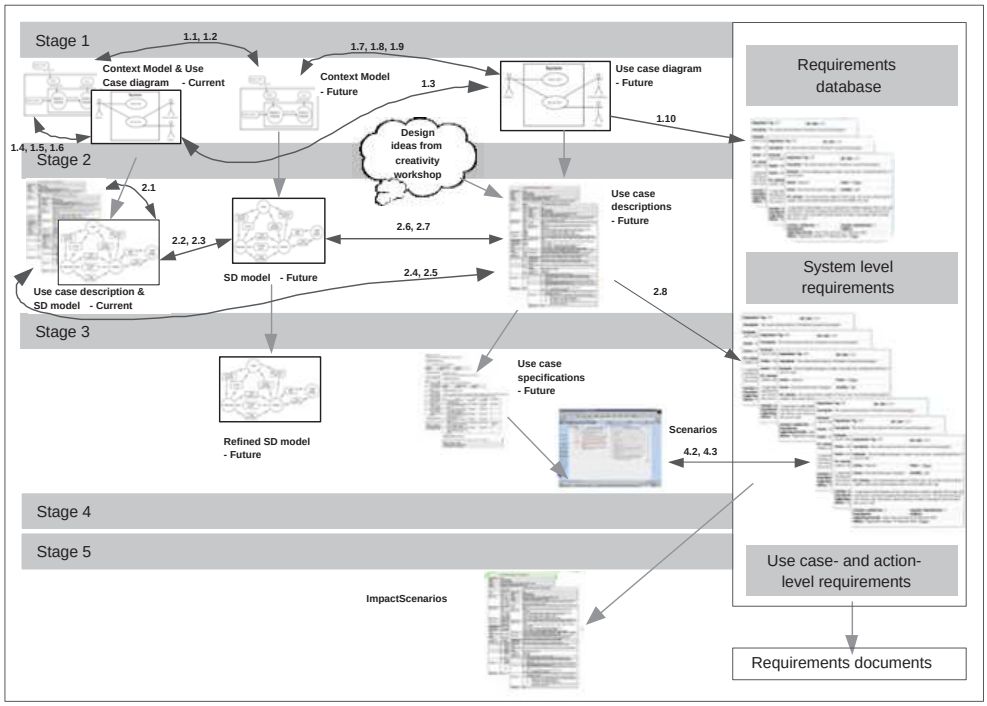

Figure 2.6: Rescue road map

2.3.1 Stage 1: the boundary stage

The first stage is called the *boundaries* stage. During this stage, information about the human processes are collected, using various techniques, like observation, informal scenario walkthroughs, interviews or contextual inquiry. The team then determines the system boundaries, by drawing con-

text models (see section 2.2.1) for the current and the future systems. Use case diagrams are build for both systems (see section 2.2.2). System level requirements identified by the activities modelled previously are described by a requirement template (see section 2.2.4) and are recorded in the requirements database in RequisitePro if their status is "Approved". They obtain this status if they pass through the quality gateway used in the RESCUE process. Other requirements are acquired using various techniques, like brainstorming, rapid prototyping or card sorting. At the end of the stage, some checks are done. "Data about the current system, as well as the (...) context models of current and future systems and the use case diagram for the current system are used to check the completeness and correctness of the use case diagram for the future system. System-level requirements are checked against use case summaries" [Maid 04c].
We give here some typical examples of checks, but all statements can be found in appendix A. All these checks will be explained in chapter 5.

Check 1.1 Every actor in the context model for the current system is a candidate actor for the context model of the future system.

Check 1.3 Every use case in the use case diagram for the current system is a candidate for a use case in the use case diagram of the future system.

Check 1.4 Every adjacent actor which communicates directly with the technical system (level 1) in the context model for the current system is a candidate actor for the use case diagram of the current system.

Check 1.6 For every data flow from or to level 1 (or one of the sub systems within level 1) of the context model of the current system, there should be a corresponding line in the use case diagram for the current system indicating involvement of the relevant actor in the relevant use case.

Check 1.10 Services and functions related to use cases in the use case diagram for the future system should map to system level requirements, i.e. high level functional and non functional requirements in the requirements database.

2.3.2 Stage 2: the work allocation stage

In the second stage, data about human activity in the current system gathered during the first stage is structured. Use case descriptions and an i* SD model (see section 2.2.3) for the current system are derived from the context model and the use case diagram built during stage 1. An i* SD model is also developed for the future system, derived from its context model.
Creative design workshops can be run to discover new requirements for the future system. These workshops take the context model, the use case diagram and some simple use case summaries providing basic descriptions of

the use cases included in the use case model as inputs. For the future system, a revised context model, new requirements and new (or revised) use cases are produced.
The use case modelling stream takes the key tasks and requirements from the use case diagram and the i* SD model along with creative design ideas and requirements from the creative design workshops, and produces use case descriptions, using the use case template.
"This is the stage at which most cross checking is done in order to bring the models of the current situation and first-cut i* models to bear on the development of correct and complete use case descriptions" [Maid 04c]. Here are some examples:

Check 2.1 All external actors in the i* SD model of the current system should correspond to actors in the use case descriptions for the current system.

Check 2.2 All external actors in the i* SD model of the current system are candidate actors for the i* SD model for the future system.

Check 2.3 All dependencies in the i* SD model of the current system are candidate dependencies for the i* SD model for the future system.

Check 2.4 All actions in the use case descriptions for the current system are candidate actions for the use case descriptions for the future system.

Check 2.7 For all task dependencies identified in the i* SD model of the future system, which represent tasks carried out by actors in the use case diagram, there should be a part of a use case description which describes how those tasks are carried out.

Check 2.8 All requirements associated with use cases using the RESCUE use case template should be stored in the requirements database.

2.3.3 Stage 3: the generation stage

In the third stage, SD model is refined and used to inform the development of more detailed use case specifications. This enables ART-SCENE, a "scenario generator" tool developed at City University, to generate a range of scenarios to be used in stage 4 of the process. Any new requirement appearing during this phase is recorded in the RequisitePro's database, using the requirement template.
No checks are needed at the end of this stage, the major output (scenarios) being generated automatically. Indeed, the requirements recorded in the RequisitePro's database will be checked at the end of the next stage.

2.3.4 Stage 4: the coverage stage

In the fourth stage, the scenarios generated during stage 3 are used to discover complete and correct requirements for the future system. They are recorded in the requirements database in RequisitePro, within the structure provided by the requirements template. These scenarios are used to drive scenario walkthroughs, using the ScenarioPresenter, which is a Web-based software tool in which access to scenarios, requirements and comments can be shared.
Once again, all the new requirements discovered during this stage have to be recorded in the requirements database, using the requirement template.
The checks at this stage relate to the structure of the requirements database. Here they are:

- Check 4.1 Ensure that each requirement is either a system level requirement or is linked to a use case, use case action, or alternative course.
- Check 4.2 Check whether each use case action is linked with requirements of the right types using simple heuristics[3] based on action- and requirement- types, for example do human-computer interaction actions have candidate functional, usability, look-and-feel, training and performance requirements specified for them.
- Check 4.3 For all except system-level requirements, check that requirement fit criteria are grounded in the use cases to which requirements are linked.

2.3.5 Stage 5: the consequences stage

The final stage is to use scenarios to analyse the impact of the future system, as it is specified in the requirements specification, on its environment. Results from this analysis can lead to changes to the requirements, which have to be recorded in the database.
The last step consists in checking if all these changes have been recorded:

- Check 5.1 Ensure that all impact consequences are recorded in the requirements database.
- Check 5.2 Ensure that change requests are generated for all impact consequences recorded in the requirements database.

[3]For more details on these heuristics, see [Maid 04b], page 110

Chapter 3

Case study: Countdown

3.1 Countdown system overview

To illustrate how the RESCUE process is used, and how the tool we developed could be useful, we chose a simple example: the Countdown system. The models, diagrams and templates found in this chapter are based on those found in [Maid 04c].
Countdown is a system designed by London buses to remove one of the biggest disadvantages of traveling by bus: uncertain waiting times. Countdown uses digital displays at bus stops to provide waiting passengers with information about waiting times. The information given by countdown displays, as found in [Jone 04], is the following:

- *The order in which buses will arrive at the stop;*
- *The number of each bus;*
- *The destination of each bus - this information originates from the driver who keys a two-digit code into the system at the start of the journey;*
- *The time until the bus arrives - based on how long the central computer estimates it will take the bus to reach the bus stop from where it is at any point in time - the precise position of the bus at any time is known from the automatic vehicle location (AVL) system;*
- *Base-line messages - the base line of the Countdown display can scroll messages across the screen from left to right every 90 seconds - messages convey general information on matters such as night buses and congestion.*

[Maid 04c] explains how the countdown system worked before its redesign:

> *"Countdown uses a network of roadside beacons to pinpoint the location of buses. These small devices are placed on lampposts along the line of the bus route and each one has a unique identity. The beacons are only activated when a bus passes and they*

transmit their identity to the bus.(. . .)

The equipment on board each bus consists of a microwave transponder, modem and odometer (wheel revolution counter). The transponder (. . .) picks up the identity of the beacons it passes and resets the odometer count. Both beacon identity and odometer count are stored in the modem which communicates through the LT[1] *Buses radio system to the central computer. A polling mechanism requests each bus location every 30 seconds.*

The central computer communicates with the network of indicators at bus stops using BT[2] *lines. It downloads information via the LT buses radio system every 30 seconds, calculates bus arrival times, and updates all Countdown indicators.*

At the start of the journey the driver keys in a code that represents the destination of the bus. The central system uses this data to determine which stops the bus will pass.(. . .) If the destination has to be changed while the bus is en-route, the driver re-keys to the new destination.(. . .)

Finally, there is a route manager for each bus route. The route manager is based at the route's garage. He uses information from the central system to manage bus routes effectively. To do this he communicates with bus drivers directly via the LT Buses radios to change routes and redirect buses."

3.2 RESCUE models and documents for the Countdown system

We will expose the models and documents produced by applying the RESCUE process for the Countdown system. Note that the models and diagrams already implements the guidelines that make them suitable to QROSS (see appendix B). Moreover, we obtained only one use case description for the current system from the RESCUE team. We built a use case description for the future system, using data we found in the requirements database. Note also that it is impossible to present the requirements database here for reasons of space. However, the latter use case description gives an idea of what it contains.

[1] London Transport
[2] British Telecom

3.2.1 Stage 1 of the RESCUE process

Context model for the current Countdown system

In this model, the technical system is composed of two subsystems: AVL system[3] and On-board bus system. The actors within level 2 -the socio-technical level- are Driver, Route controller and Indicator. In level 3, we find Road-side beacon, Passenger, London Transport and Communication System. There is no level 4.

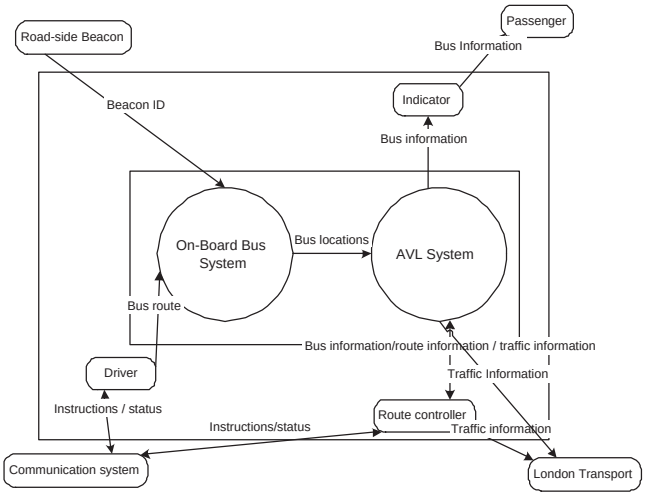

Figure 3.1: Context model for the current Countdown system

Use case diagram for the current Countdown system

In the use case diagram for the current system, we find the same actors as those directly communicating with the level 1 in the context model, i.e. Driver, Indicator, Road-side beacon, Route controller and London Transport. Passenger and Communication system do not communicate directly with the level 1 in the context model and are not represented in the use case diagram modelling these interactions.

[3]Automatic Vehicle Location system

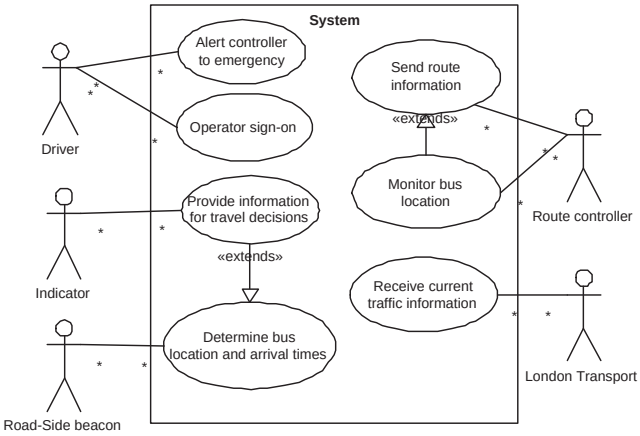

Figure 3.2: Use case diagram for the current Countdown system

Context model for the future Countdown system

Within level 1, we find the systems to be redesigned, composed of two subsystems: AVL System and On-board bus system. All the other actors are the same as in the context model for the current system (figure 3.1), except that Indicator has been replaced by Display and Road-side beacon by GPS. The boundaries have been refined, if we compare with the context model for the current system: The passenger is included within level 3, and a level 4 has appeared, including Communication system, London Transport and GPS.

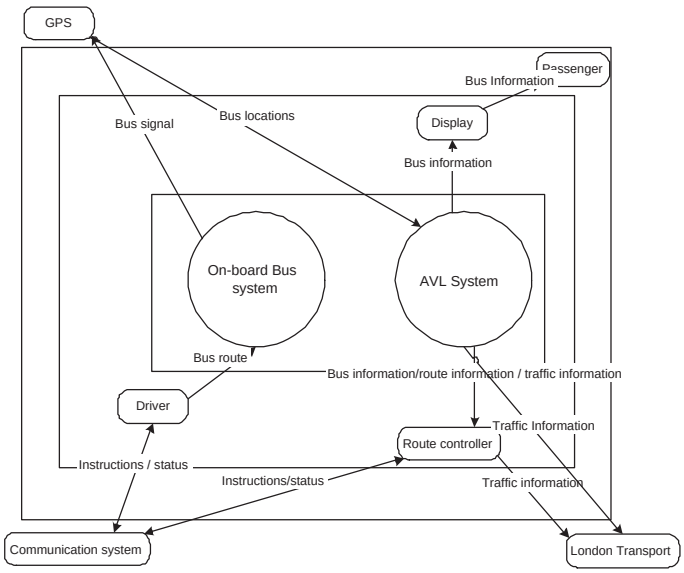

Figure 3.3: Context model for the future Countdown system

Use case diagram for the future Countdown system

Here again, the actors represented are the actors communicating directly with level 1 in the context model. Communication system and Passenger are thus not represented.

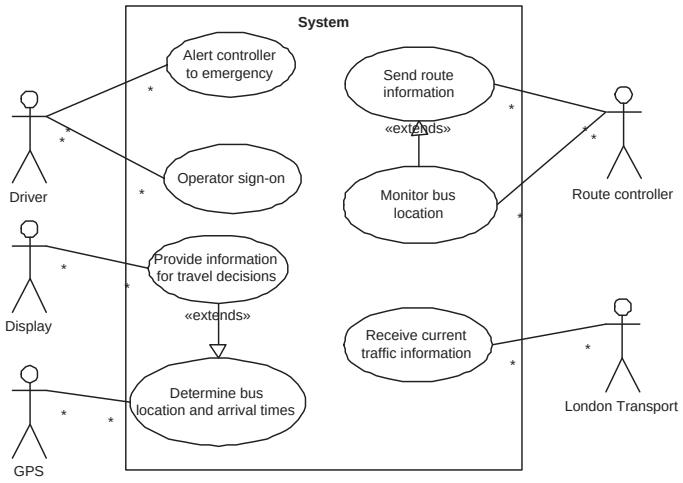

Figure 3.4: Use case diagram for the future Countdown system

3.2.2 Stage 2 of the RESCUE process

SD model for the current Countdown system

From the context model and the use case diagram for the current system (see respectively figures 3.1 and 3.2), an i* SD model is built for the current system, modelling goals, rationale and dependencies of the system. The actors marked with a "*" are non external actors and correspond to actors within level 1 and 2 of the context model.

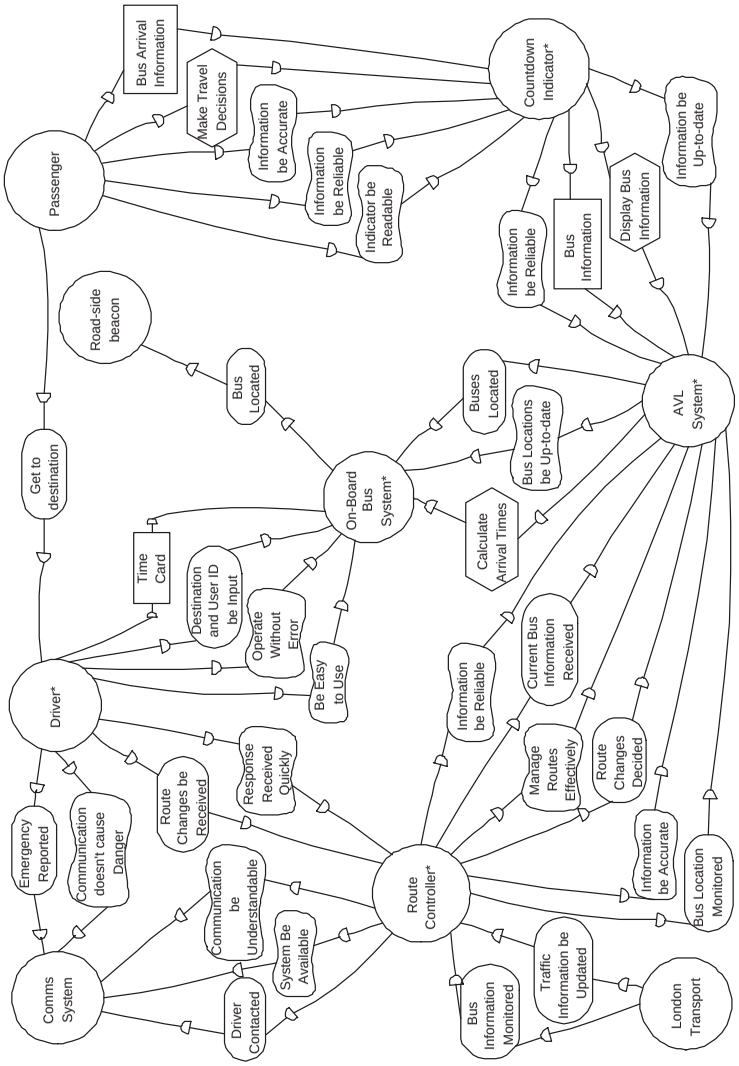

Figure 3.5: SD model for the current Countdown system

Use case descriptions for the current Countdown system

As we said in the introduction, we obtained only one use case description for the Countdown system. It is the use case description corresponding to the use case "Provide information for travel decisions" related to the actor "Indicator".

Use case ID	UC5 Current
Name of use case	Provide information for travel decisions
Text	/
Author	J. Bloggs
Date	12^{th} February 2003
Source	Observation and interviewing at a bus stop in Holloway Road
Actors	Indicator, Passenger, AVL system
Problem statement (now)	No problemo
Precis	The indicator at the bus stop provides information for passengers to make travel decisions.
Functional requirements	The same information about bus arrival times should be available on all types of display in the future system. Information about which buses are mobility buses should be displayed on the Countdown indicator.
Non-functional requirements	Estimated bus arrival times should be accurate to within 2 minutes.
Added value	Undefined
Justification	Undefined
Triggering event	Passenger seeks bus information from the Countdown indicator.
Preconditions	Passenger is at bus stop. Countdown indicator is present at bus stop.
Assumptions	Undefined
Successful end states	Passenger gets enough information about bus arrival times to make a satisfactory decision about travel.
Unsuccessful end states	Passenger gets insufficient information to make a satisfactory travel decision. Passenger does not understand information presented.
Different walk-through contexts	Undefined

Table 3.1: Use case description for the current Countdown system (part 1)

Normal course	1. The passenger seeks bus information from the Countdown indicator.
	2. The Countdown indicator shows the bus information for the relevant route(s).
	3. The passenger reads bus information from the Countdown indicator.
	4. The passenger recognises which route number(s) will take them closest to their destination.
	5. The passenger remembers expected arrival time(s) for bus(es) on route(s) of interest.
	6. The passenger uses the bus information to make decisions about their journey.
	7. Every 30 seconds the AVL system updates the Countdown indicator.
	8. The passenger occasionally checks his/her decision when information on the indicator is updated.
Variations	1. If passenger has a mobility restriction, then passenger seeks information about mobility buses from the Countdown indicator.
	6. If wet weather, then passenger may decide not to use a bus if expected waiting time is too long
	6. If night, then passenger may decide not to use a bus if expected waiting time is too long
Version history	Maintained by RequisitePro
Satisfaction arguments	Undefined
Option	Undefined
Features	Undefined

Table 3.2: Use case description for the current Countdown system (part 2)

SD model for the future Countdown system

From the context model and the use case diagram for the future model (see figures 3.3 and 3.4 respectively), an i* SD model is derived. The non external actors in this SD model are marked by a "*" and correspond to the actors within levels 1 and 2 in the context model for the future system.

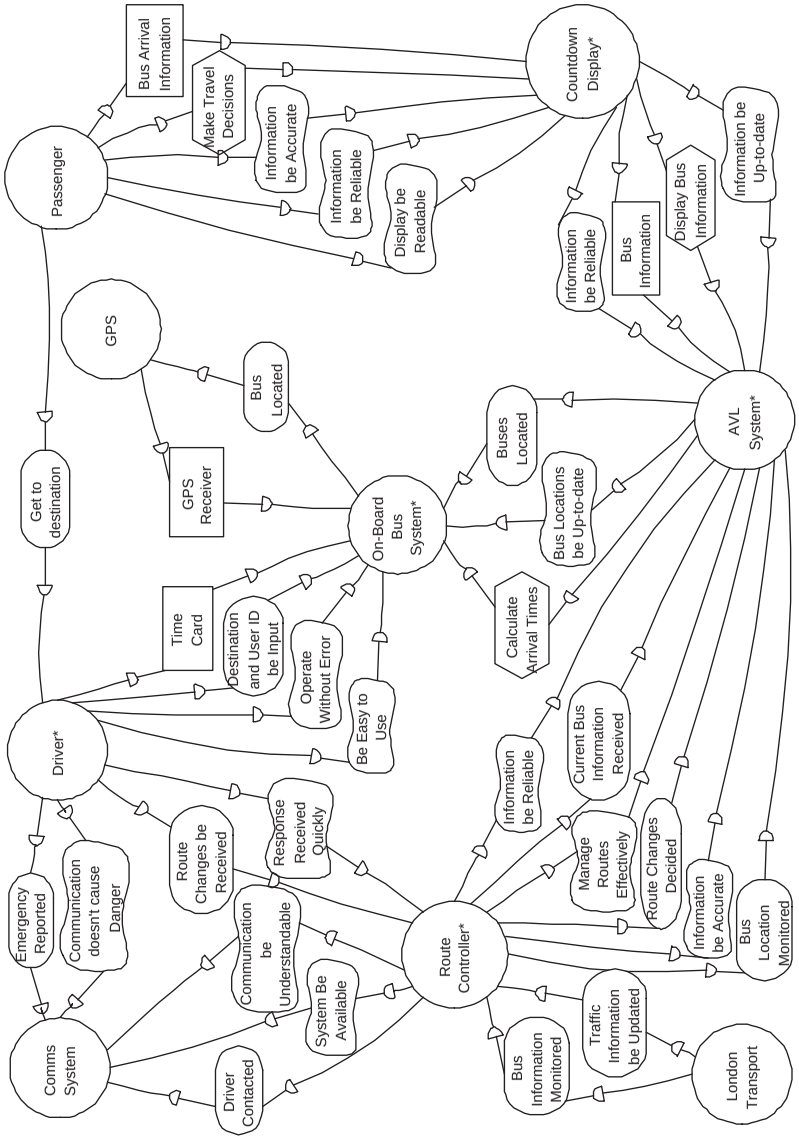

Figure 3.6: SD model for the future Countdown system

Use case description for the future Countdown System

As we already said, this use case description has been built using data we found in the requirements database. The database we had access to was incomplete, and thus, the information provided in this description is not complete either.

Use case ID	UC1 Future
Name of use case	Provide information for travel decisions
Text	/
Author	Bloggs
Date	2^{nd} December 2003
Source	interviews with project managers
Actors	
Problem statement (now)	No problemo
Precis	The display at the bus stop provides information for passengers to make travel decisions.
Functional requirements	Undefined
Non-functional requirements	Undefined
Added value	Undefined
Justification	Undefined
Triggering event	Passenger seeks bus information from the Countdown indicator.
Preconditions	Passenger is at bus stop. Countdown display is present at bus stop.
Assumptions	Undefined
Successful end states	Passenger gets enough information about bus arrival times to make a satisfactory decision about travel.
Unsuccessful end states	Passenger gets insufficient information to make a satisfactory travel decision. Passenger does not understand information presented.
Different walk-through contexts	Undefined

Table 3.3: Use case description for the future Countdown system (part 1)

Normal course	1. The passenger looks at the Countdown display.
	2. The Countdown display shows the bus information for the relevant route(s).
	3. The passenger recognises which route number(s) will take them closest to their destination.
	4. The passenger remembers expected arrival time(s) for bus(es) on route(s) of interest.
	5. The passenger decides which bus route to use.
	6. Every 30 seconds the AVL system transmits updated bus information to the Countdown display.
	7. The passenger occasionally checks his/her decision when information on the indicator is updated
Variations	1. If the passenger has poor eyesight, then he can seek information from the Countdown display in an audible way.
	6. If the AVL system does not update the various Countdown displays with the new expected arrival times, then a passenger message is displayed on the Countdown displays.
Version history	Maintained by RequisitePro
Satisfaction arguments	Undefined
Option	Undefined
Features	Undefined

Table 3.4: Use case description for the future Countdown system (part 2)

Chapter 4

Requirements and architecture of the QROSS-Checker

4.1 Requirements

The aim of our work is to automate the synchronisation checks of the RESCUE process. These checks are indeed great time spendings. Their automation is meant to spare RESCUE users time and money.
We took more interest in the first two phases for the logical reason of chronology, first. Another reason is that the first phase is quite simple, so it makes a good start for the understanding of the checking process. The second phase is the one which takes more time to apply manually. Its automation will thus be the most rewarding.

Hence, the idea is to build a tool which takes the models developed in the two first phases as input and produces a text file containing the results of the checks as output, as shown on figure 4.1.

4.2 QROSS: description

QROSS is a web based application, coded in Python, developed by the CETIC. It was initially designed to compute quality metrics on source code. To this end, QROSS imports code from two kinds of sources: file systems and CVS servers. The code is then parsed, using source navigator, in order to populate the tool's object oriented repository, based on ZODB. The metrics are computed on the artefacts contained in this repository. They are expressed in OCL, the Object Constraint Language which is part of the UML ([Rumb 99]). We can see the architecture of the tool on figure 4.2, and the structure of the repository on figures 4.3 and 4.4.

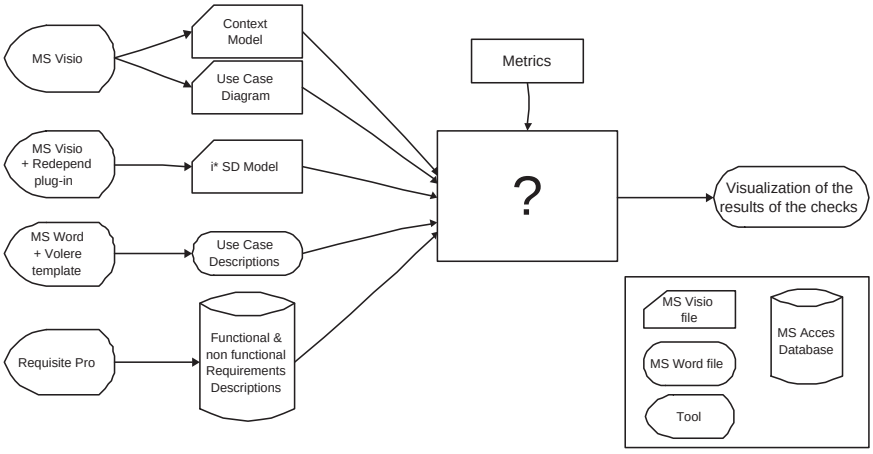

Figure 4.1: Inputs and outputs for the RESCUE process.

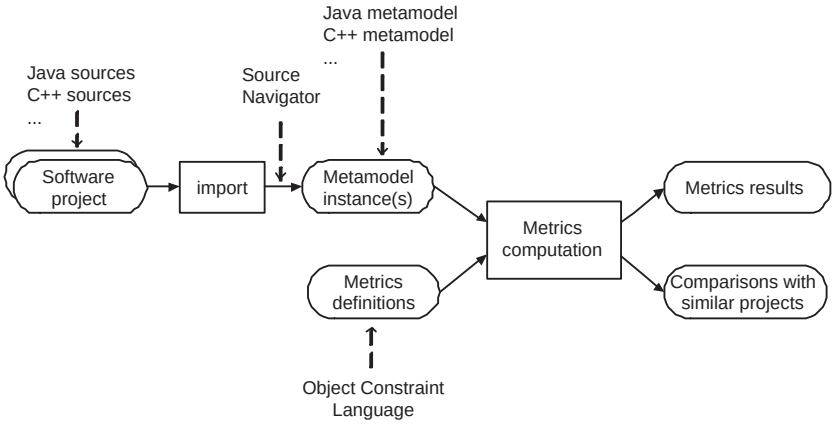

Figure 4.2: Architecture of the tool QROSS.

The tool source navigator is dedicated to parse only source code. Moreover, at that time, the repository could contain Java or C++ source code but nothing else; the class "*ModelElement*" found in the figure 4.3 was not elaborated further in the current version of the tool. Thus, we had to extend QROSS to make it suitable for dealing with RESCUE related artefacts, by implementing parsers and adapting the repository.

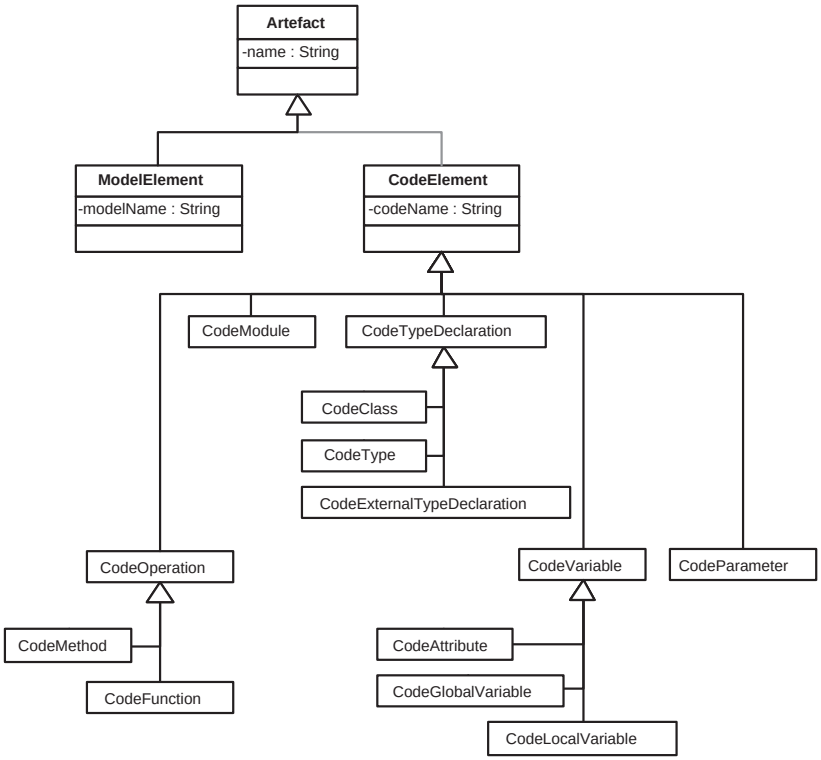

Figure 4.3: General structure of the repository.

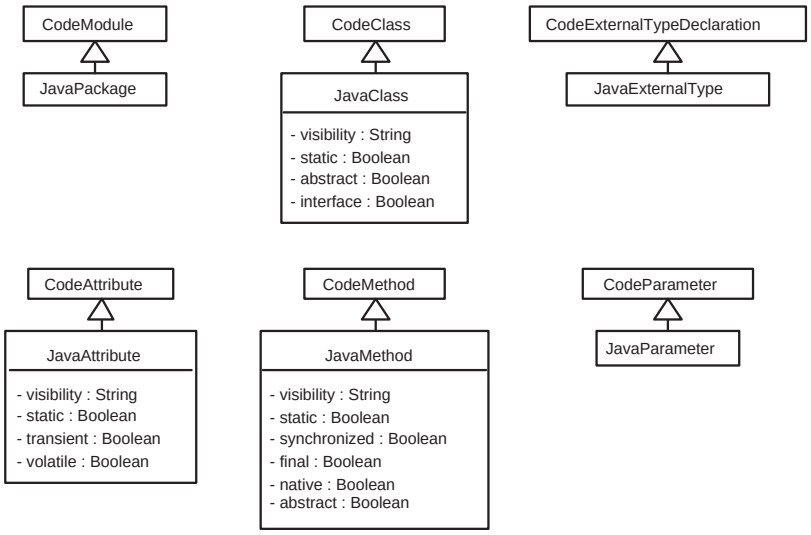

Figure 4.4: Structure of the repository specific to Java code.

4.3 Architecture of the QROSS-Checker

Our checker was built by extending QROSS. Metrics are now computed on other types of artefacts, taken from the meta models of the various notations and templates used in the RESCUE process.
The files produced during the different phases of the RESCUE process, recorded using formats suitable to QROSS, are the inputs of QROSS. We implemented parsers and repository loaders for each of these files. By parsing these files, we generate a database content representing the various inputs. Finally comes the stage of the checks. The greatest part of these is done by using OCL requests made on the elements of the generated database content. Figure 4.5 represents this process.

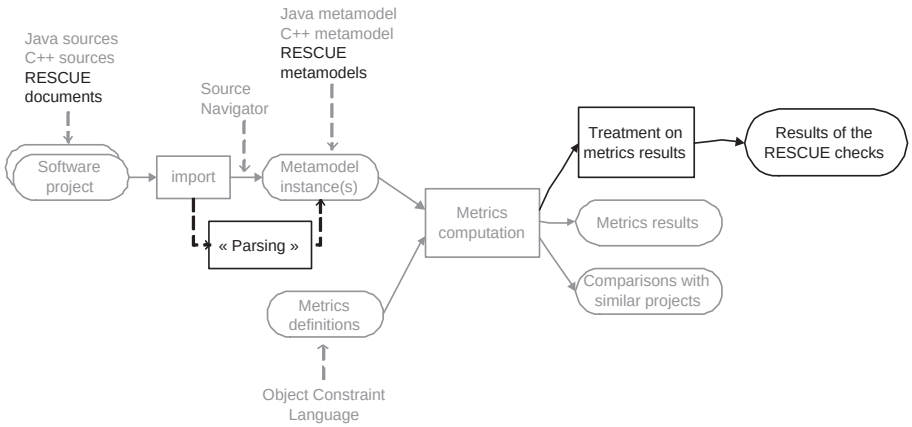

Figure 4.5: Architecture of our tool, by extending and adapting the architecture of QROSS.

4.4 Repository of the QROSS-Checker

To prepare the implementation to populate the repository, we built a meta model for each concept encountered in the RESCUE process. We will describe those in the following subsections, in their order of appearance in the RESCUE process. An explanation on the details of importance for the remainder of this thesis will be joined to each schema.

4.4.1 Conceptual meta models

Meta model for RESCUE projects

Firstly, we meta modeled what a RESCUE project should contain.

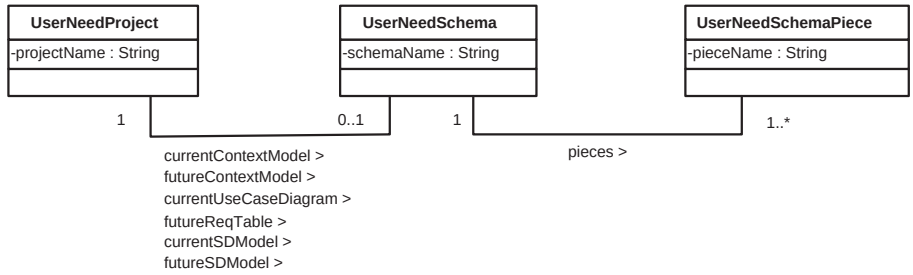

Figure 4.6: Meta model for RESCUE projects

On figure 4.6, there should be one relationship for each type of schema related to a project but, since they are all similar, we drew only but repeated the name of the relationship.

So, we can see on figure 4.6 that a RESCUE project can contain one context model for the current system, one for the future system, one use case diagram for the current system, one for the future system, one requirements database (for the future system), one SD model for the current system and one for the future system.
To be significant, a RESCUE project must not be empty, i.e. it must contain at least one schema. Each of these schemas is composed of at least one piece.
A *UserNeedSchema* is identified by a *schemaName*, and a *UserNeedSchemaPiece* by a *pieceName*.

In the following subsections, we describe the meta model for each type of schema.

Meta model for context models

The first model appearing in the RESCUE process is the context model. Figure 4.7 shows its meta model.
As we can see on that figure, a context model schema is composed of agents and of informations exchanged between those agents. To be significant, a context model must contain at least one agent.

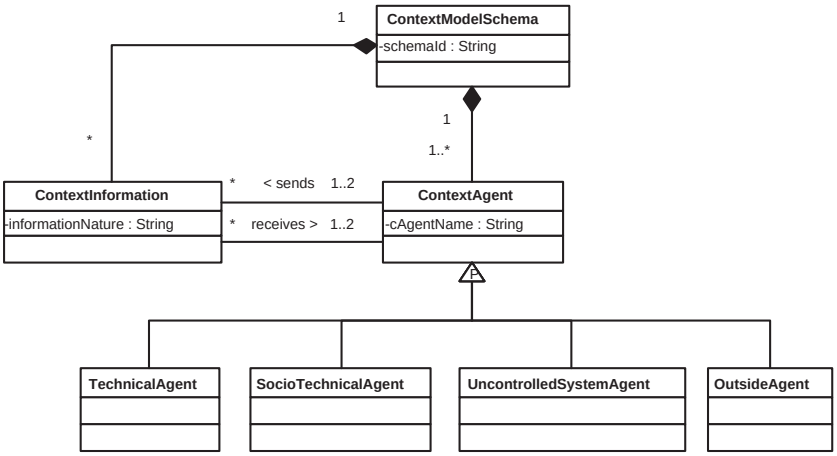

Figure 4.7: Conceptual meta model for context models

An agent can send and receive several types of information. An agent must be a technical agent, a socio technical agent, an uncontrolled agent, or an outside agent.

An information is sent by one and only one agent and received by one and only one agent in the case of a unidirectional data flow, and sent and received by two and only two agents in the case of a bidirectional data flow.

The attribute *schemaId* in the class *ContextModelSchema* represents the name given to the MS Visio file containing the schema. The attribute *cAgentName* in the class *ContextAgent* represents the name given to the actor on the schema. Finally, the attribute *informationNature* in the class *ContextInformation* represents the name we found on the arrow representing this flow of information in the schema.

Meta model for use case diagrams

As we can see on figure 4.8, a use case diagram is composed of actors and of use cases. To be significant, a use case diagram must contain at least one actor.

An actor can be involved in various use cases, and several actors can be involved in the same use case.

The attribute *ucdName* in the class *UseCaseDiagram* represents the name given to the MS Visio file containing the diagram. The attribute *actorName* in the class *UseCaseActor* represents the name given to this actor on the schema. The attribute *ucName* in the class *UseCase* represents

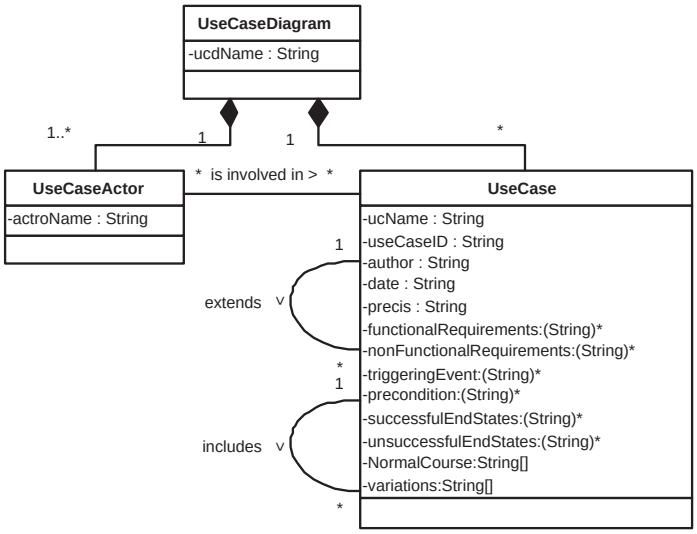

Figure 4.8: Conceptual meta model for use case diagrams

the name given to the use case on the schema. All the other attributes are the ones we find in the use case description corresponding to the use case (see table 2.1 on page 11). The relation between a use case found on the use case diagram and its use case description is done using the attribute *ucName*, which has to be the same as the name of the use case found in the use case description.

Meta model for SD models

We can see on figure 4.9 that an SD model is composed of actors and of dependencies between those actors. To be significant, an SD model must contain at least one actor.

An actor can be the depender or the dependee for several dependencies.

A dependency is related to one and only one actor being the depender of it, and to at least one actor being the dependee for it[1]. An SD dependency is either a resource dependency, a task dependency, a goal dependency or a soft goal dependency.

As usual, the attribute *sdName* in the class *StrategicDependencyModel* represents the name given to the MS Visio file containing the schema. The attribute *actorName* in the class *SDActor* represents the name given to

[1] Although that is not frequently seen, several dependees for a dependency are allowed, to solve problems of space in big SD model diagrams

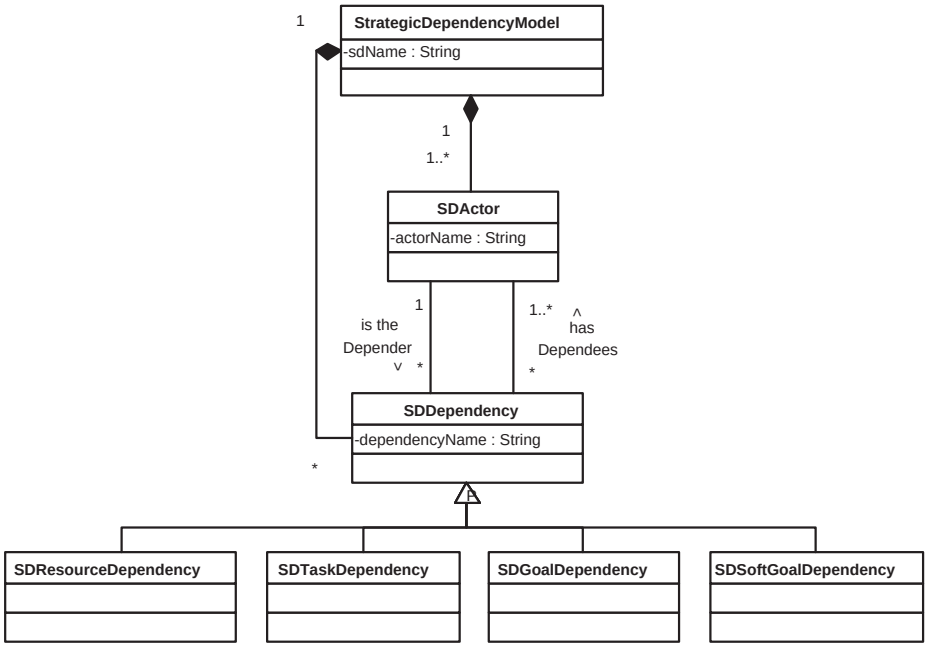

Figure 4.9: Conceptual meta model for SD models

the actor on the schema, and the attribute *dependencyName* in the class *SDDependency* represents the name given to the dependency on the schema.

Meta model for requirements databases

The meta model for requirements databases is shown on figure 4.10. A requirements database is composed of requirements. To be significant, a requirements database must contain at least one requirement.

The attributes in the class *Requirement* are the ones found in the requirement template used in RequisitePro (see table 2.2 on page 14). As there is only one requirement table, it is not necessary to give it a name to identify it. That is the reason why there is no attribute *name* in the class *RequirementsTable*.

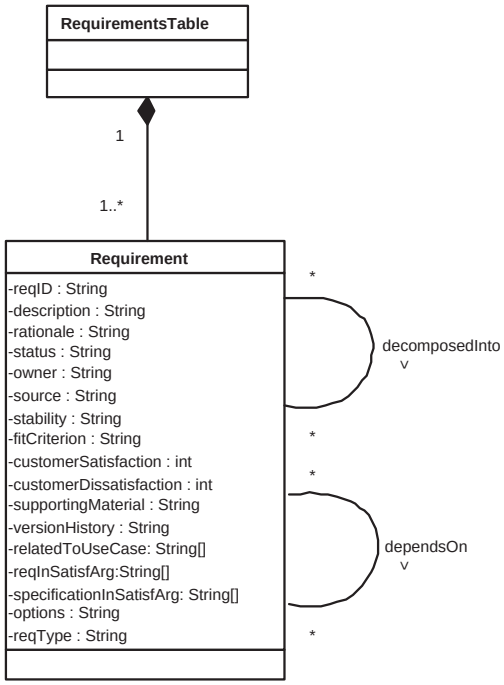

Figure 4.10: Conceptual meta model for requirements databases

4.4.2 Implementing the meta models

We will introduce this section with an explanation about notations used in the schemas we show in the following subsections.

(type)* **:** describes a list of elements of type *type*. The list can be empty.

(type)+ **:** describes a list of elements of type *type*. The list contain at least one element.

- : the attribute or method following this symbol is a private one.

+ : the attribute or method following this symbol is a public one.

General repository

The repository of the QROSS-Checker needed to be extended to be able to contain the models being part of the RESCUE process. We can see these extensions on figure 4.11.

The first thing to explain about the choices of implementation we made is the reason why we do not use the class *ModelElement* provided in the former repository of QROSS. As we have seen on figure 4.6, we are working

in terms of projects. However, a RESCUE project is not a model, it is a set of models. So, we chose to create a new class, inheriting the class *Artefact*, *UserNeedElement*.

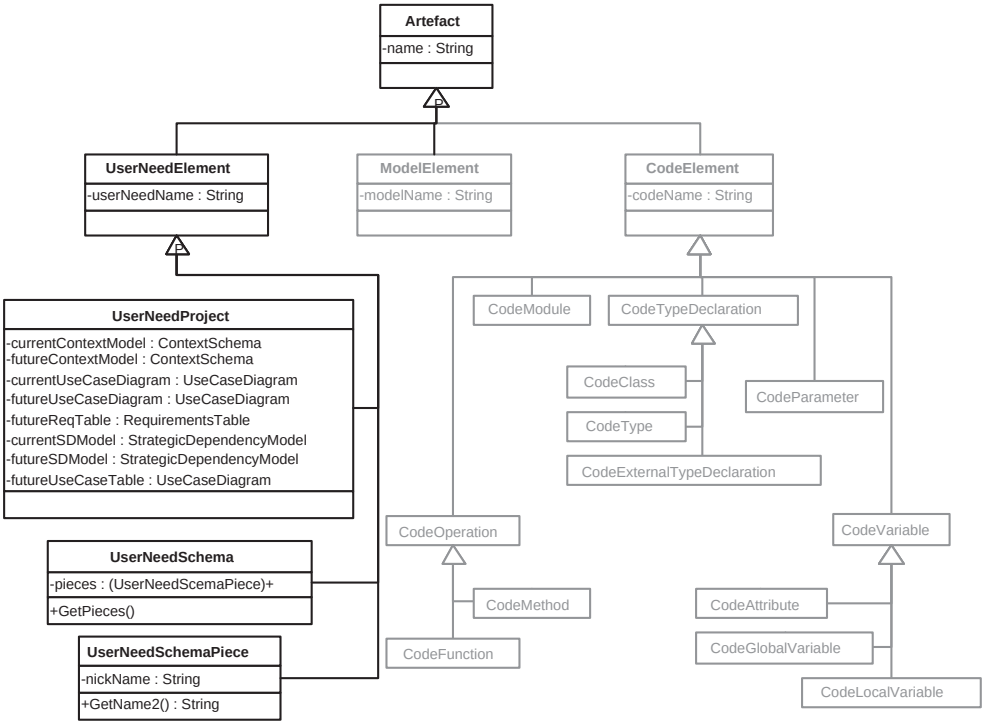

Figure 4.11: Repository of the QROSS-Checker

As we noticed that the various classes in the meta model for RESCUE projects on figure 4.6 contained a similar attribute "name" containing a string, we decided to put it on the *UserNeedElement* class. Indeed, the classes *UserNeedProject*, *UserNeedSchema* and *UserNeedSchemaPiece* inherit this class and thus its attributes.

The relation *pieces* between *UserNeedSchema* and *UserNeedSchemaPiece* was transformed into a *UserNeedSchema* attribute containing a list of at least one element of type *UserNeedSchemaPiece*.
We added an attribute *nickName* in *UserNeedSchemaPiece*. This will be used for pieces of schemas drawn with MS Visio. For these elements, the attribute *userNeedName* in the class *UserNeedElement* contains the figure identifying the shape in the MS Visio file. The attribute *nickName* contains the name given to the shape in the MS Visio file. The method *GetName2()* returns the *nickName*. This will be useful in implementing the checks.
Finally, the relations between *UserNeedProject* and *UserNeedSchema* found

on the figure 4.6 were transformed into *UserNeedProject* attributes containing an element of a subclass of *UserNeedSchema*. An additional attribute, "futureUseCaseTable", has been added. It is used to store the use case descriptions found in the RequisitePro database. They are indeed two sources for MS Word documents containing the use case descriptions: MS Word files and entries in the RequisitePro data base. We need to be able to compare the two of them (see check 2.8), so we decided to keep both in separate tables in our repository. This will be useful to implement the checks 2.4 and 2.5 (see sections 5.3.4 and 5.3.5). It contains a *UseCaseDiagram* with instances of *UseCase* only, no instance of *UseCaseActor*. This ignores the constraint of integrity saying that a use case diagram is significant only if it contains at least one instance of *UseCaseActor*. But, the content being the same for use cases coming from use case descriptions in an MS Word format and use case descriptions found in the RequisitePro database, it was not useful to implement a new model.

Each schema will be a subclass of the class *UserNeedSchema* and each piece of those schemas will be a subclass of the class *UserNeedSchemaPiece*. These subclasses will be detailed in the following sections.

Repository specific to context models

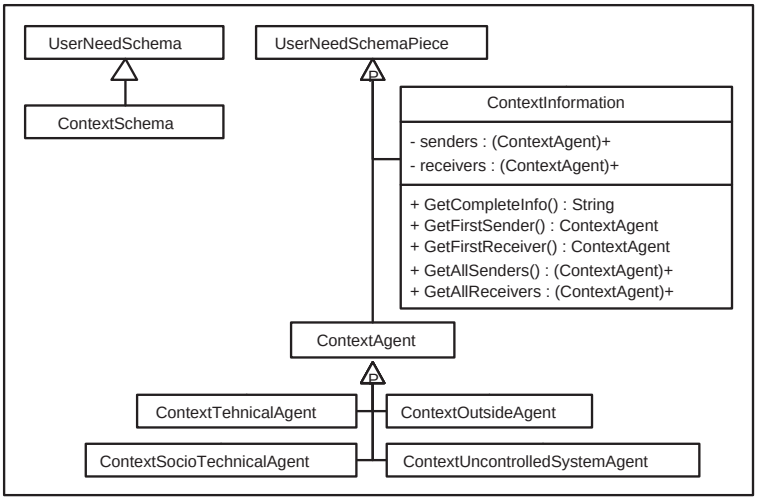

Figure 4.12: Repository specific to context models

As we said above, the class *ContextSchema* inherits the class *UserNeedSchema*. Its two types of pieces, *ContextAgent* and *ContextInformation*, inherit the class *UserNeedSchemaPiece*. The attributes *schemaId* in *ContextSchema*, *cAgentName* in *ContextAgent* and *informationNature* in *Con-*

textInformation[2] are now recorded in the attribute *userNeedName* in *UserNeedElement*, as all those classes inherit the latter one.

The relations "sends" and "receives" between *ContextAgent* and *ContextInformation* became attributes containing *ContextAgent* in *ContextInformation*. This choice has been done because having the name of the information, its senders and its receivers is the only way to identify a *ContextInformation*. Indeed, the same information can be sent by various agents, and an information sent by an agent can be received by various agent. Let us notice that the attributes "senders" and "receivers" will contain at least one (for unidirectional data flows) and at most two (for bidirectional data flows) elements of the type *ContextAgents*. In a given instance of *ContextInformation*, both lists have the same length (1 or 2).

The method *GetCompleteInfo()* on *ContextInformation* objects returns a string containing the name of the information exchanged and the name of its senders and receivers, separated by special characters.
The methods *GetFirstSender()* and *GetFirstReceiver()* on *ContextInformation* objects returns an element of type *ContextAgent* containing the object representing the first sender found in the attribute *senders* and the first receiver found in the attribute *receivers* respectively. This will be used in the implementation of the checks. Let us point out that those methods, applied on the same element, always return different instance of *ContextAgent*, even if the data flow is a two-way one. Indeed, if a two-way data flow takes place between an agent "A" and an agent "B", we have the following property:

- the attribute *senders* will contain
  ["*ContextAgent* instance "A"", "*ContextAgent* instance "B""]
  (or ["*ContextAgent* instance "B"", "*ContextAgent* instance "A""]);

- the attribute *receivers* will contain
  ["*ContextAgent* instance "B"", "*ContextAgent* instance "A""]
  (or ["*ContextAgent* instance "A"", "*ContextAgent* instance "B""] respectively).

A call to those methods on an object *ContextInformation* having this property allows us to determine the agents between which the data flow takes place without having to compare the agents collected.
The methods *GetAllSenders()* and *GetAllReceivers()* on *ContextInformation* objects simply returns a list of at least one and at most two elements of type *ContextAgent* containing the objects representing the senders found in the attribute *senders* and the receivers found in the attribute *receivers* respectively.

[2] see figure 4.7 for recall

Repository specific to use case diagrams

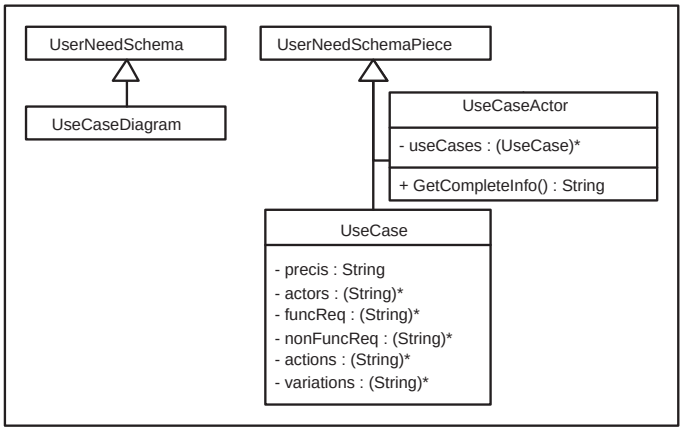

Figure 4.13: Repository specific to use case diagrams

As we said, the class *UseCaseDiagram* inherits the class *UserNeedSchema*. Its two types of pieces, *UseCaseAgent* and *UseCase*, inherit the class *UserNeedSchemaPiece*. The attributes *ucdName* in *UseCaseDiagram*, *actorName* in *UseCaseActor* and *ucName* in *UseCase*[3] are now recorded in the attribute *userNeedName* in *UserNeedElement*, as all those classes inherit the latter one.

The relation "is involved in" between a *UseCaseActor* and a *UseCase* has been implemented as an attribute *useCases* containing a list of *UseCases*. This list can be empty, hence an actor is not necessarily involved in any use case.

The method *GetCompleteInfo()* on *UseCaseActor* objects returns a string containing the name of the actor and the use cases he is involved in, separated by special characters.

Some relations and attributes are present in the class *UseCase* on figure 4.8 but not on figure 4.13. This is due to the fact that, after analysing the checks and discussions with the RESCUE team, a few fields of the use case template (see table 2.1 on page 11) were useful to implement the checks, which is the aim of our work. The relevant ones are the followings :

- Use case ID;
- Name of use case;
- Actors;

[3]see figure 4.8 for recall

- Precis;
- Functional requirements;
- Non functional requirements;
- Normal course;
- Variations.

Repository specific to SD models

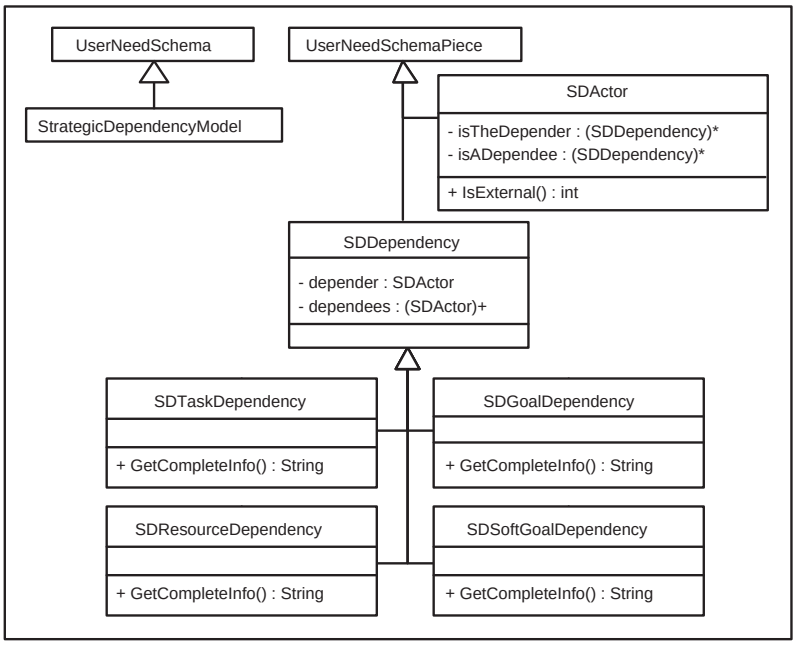

Figure 4.14: Repository specific to SD models

As usual, the class *StrategicDependencyModel* inherits the class *UserNeedSchema*. Its two types of pieces, *SDActor* and *SDDependency*, inherit the class *UserNeedSchemaPiece*. The attributes *sdName* in *StrategicDependencyModel*, *actorName* in *SDActor* and *dependencyName* in *SDDependency*[4] are now recorded in the attribute *userNeedName* in *UserNeedElement*, as all those classes inherit the latter one.

The relations "is the Depender" and "has Dependees" between *SDActor* and *SDDependency* became attributes in both *SDActor* and *SDDependency* classes. The redundancy of this information is justified because it increases the effectiveness of the implementation of the checks.

[4]see figure 4.9 for recall

The method *GetCompleteInfo()* on each type of *SDDependency* objects returns a string containing the type of the dependency, its name, its depender and its dependees, separated by special characters.
The method *IsExternal()* on *SDActor* objects returns "1" if the actor is an external one, "0" if not.

Repository specific to requirements tables

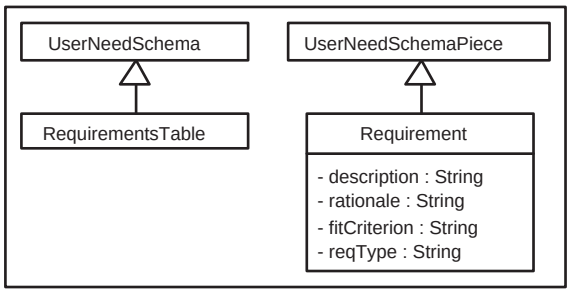

Figure 4.15: Repository specific to requirements tables

The class *RequirementsTable* is a subclass of *UserNeedSchema*, and *Requirement* is a subclass of *UserNeedSchemaPiece*.
So, as usual, the attribute *reqID* in *Requirement* has been removed, the class inheriting indirectly the class *UserNeedElement* and its attribute *userNeedName*. The attributes appearing on figure 4.10 but not on figure 4.15 have not been implemented because they are of no interest for the checks.

4.4.3 OCL interpreter

To implement the checks, we use metrics expressed in OCL. The OCL interpreter included in QROSS have an access to all the classes we defined above and to their public and private attributes and methods. Moreover, it can access attributes and methods inherited from QROSS classes, or from Python classes on top of which they are built. In particular, we will often make use of the method "*myobject.__class__.__name__*" which provides us with the name of the class the object *myobject* is an instance of.

We give here a short introduction to the Object Constraint Language, announcing the major elements of syntax which will bring the reader to a better understanding of the remainder of this thesis, especially in chapter 5. "*The Object Constraint Language (OCL) is a text language for writing navigation expressions for constraints, guard conditions, actions, preconditions and post-conditions, assertions and other kinds of UML expressions.*" [Rumb 99]

"*Syntax for some common navigation expressions is shown below. These forms can be chained together. The left-most element must be an expression for an object or a collection of objects. The expressions are meant to work on collections of values when applicable.*" [Rumb 99]

item.selector selector is the name of an attribute in the name of the item or the name of a role of the target end of a link attached to the item. The result is the value of the attribute or the related object(s).

item.selector[**argument-list**] selector is the name of an operation on the item. The result is the return value of the operation applied to the item.

collection→collection-property collection-property is the name of a built-in OCL function on collections. The result is the property of the collection. Illegal if the collection property is not a predefined OCL function. Several of the properties are listed below.

collection→select(boolean-expression) boolean-expression is written in terms of objects within the collection. The result is the subsets of objects in the collection for which the expression is true.

collection→size The number of elements in the collection.

self Denotes the current object (may be omitted if the context is clear).

operator The usual arithmetic and Boolean operators: =, <, >, <=, >=, <>, +, -, *, /, not.

We give here a simple example relative to the class diagram of figure 4.16:
School.students→select(result = 'failed')→size
This query counts the number of instances of Student attached to the current instance of School who have failed.

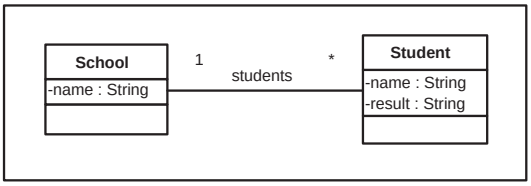

Figure 4.16: Example for OCL: class diagram

4.5 Parsers

To make it easier to implement parsers, we had to constrain the users to use the existing tools in a defined way.
Firstly, the context model and the use case diagrams were originally drawn using MS Word. This turned out to be very complex to parse, due to the unforeseeable and complex format of the files produced. Hence, we decided, in dialogue with the RESCUE team, to constraint the users to build it with MS Visio. This tool being already used to draw the SD model, this also limits the number of different tools used in the RESCUE process. We defined the shapes to use, and the correct way to draw the models and diagrams. This is translated into a set of guidelines reproduces in appendix B on page 115. An example of the code for parsers can be found in appendix D on page 121.

Chapter 5

Implementing the checks

5.1 Introduction

In this chapter, we explain the reasoning made to implement the checks. Each of the following section is devoted to a particular check and follows the same template. We first transcribe the original statement of the check found in [Maid 04c]. Then, if needed, we clarify its meaning and remove possible ambiguities. Clarifications were made by asking the RESCUE team for the intended meaning of the checks. We also explain the difficulties we have met to properly understand and implement the check. We then provide its implementation in OCL, sometimes completed with custom python code when needed. Finally, the MS Word table generated which is used to question the RESCUE stakeholders is shown. Sometimes, an example is provided to make the concepts clearer. When the code is not provided, it can be found in appendix E.

5.2 First phase of synchronisation checks

5.2.1 Check 1.1

Original statement

This check was defined in the RESCUE process as follow:

> *Every actor in the context model for the current system is a candidate actor for the context model of the future system.*

Precise statement

We transcribed this as follow:

> *For each actor found in the context model for the current system but not found in the context model for the future system, are we*

sure we want to remove it from the context model for the future system?
For each actor found in the context model for the future system but not found in the context model for the current system, are we sure we want to add it to the context model for the future system?

Difficulties

The main difficulty here was the use of the word "candidate". We had to define it precisely to find exactly what the users of the RESCUE process want to have as a result.
The first idea was that all the actors present in the current model had to be in the future model, i.e. the context model for the future system contained at least all the actors of the model for the current system.
After discussions with the users, we found that this was not exactly the concept. It is more about the fact of being sure whether one wants to remove or add an actor in the model.
So, the idea is to produce two lists. The first contains the actors present in the current system but not in the future system. The second one is the list of the actors present in the future system but not in the current one. Questions will be asked to the stakeholders based on these lists.

Implementation

This has been implemented using only the OCL language. It is indeed quite simple.
The first thing to do was to select all the actors of each model. This has been done using the two following queries.

```
CurrentContextAgents = currentContextModel.pieces→
        select (agent | agent.__class__ .__name__= 'ContextTechnicalAgent'
        or agent.__class__ .__name__= 'ContextSocioTechnicalAgent'
        or agent.__class__ .__name__= 'ContextOutsideAgent'
        or agent.__class__ .__name__= 'ContextUncontrolledSystemAgent')
        .GetName2()
FutureContextAgents = futureContextModel.pieces→
        select (agent | agent.__class__ .__name__= 'ContextTechnicalAgent'
        or agent.__class__ .__name__= 'ContextSocioTechnicalAgent'
        or agent.__class__ .__name__= 'ContextOutsideAgent'
        or agent.__class__ .__name__= 'ContextUncontrolledSystemAgent')
        .GetName2()
```

We then made the disjunction between these two sets of actors. Here are the OCL requests used to produce the result.

```
Check_1_1_Part1 = CurrentContextAgents→
                  select (actor | not FutureContextAgents→
                  includes(actor))
Check_1_1_Part2 = FutureContextAgents→
                  select (actor | not CurrentContextAgents→
                  includes(actor))
```

The elements of the lists "Check_1_1_Part1" and "Check_1_1_Part2" are then included in the MS Word table containing the result of the check (see table 5.1). The table will then be included in the MS Word file generated by the QROSS-Checker.

MS Word table

Check 1.1 *Every actor in the context model for the current system is a candidate actor for the context model of the future system.*		
Do you want to remove these actors to create your new system?		
	Comments from City	Comments from others
Output_1_1_Part1_1		
Output_1_1_Part1_2		
...		
Do you want to add these actors to create your new system?		
	Comments from City	Comments from others
Output_1_1_Part2_1		
Output_1_1_Part2_2		
...		

Table 5.1: Table containing the results of the check 1.1

5.2.2 Check 1.2

Original statement

> *Every data flow in the context model for the current system is a candidate data flow for the context model of the future system.*

Precise statement

> *For each data flow found in the context model for the current system but not found in the context model for the future system, are we sure we want to remove it from the context model for the future system?*
> *For each data flow found in the context model for the future system but not found in the context model for the current system, are we sure we want to add it to the context model for the future system?*

Difficulties

The difficulty found here is the same as in the check 1.1, i.e. the use of the word "candidate". The reasoning is thus the same as in 5.2.1, applied to data flows instead of actors.

Implementation

The first step is to collect all the data flows we find in both context models in order to produce two lists, each representing the data flows present in its respective context model. Here are the OCL queries used to do this:

```
CurrentDataFlowInfo = currentContextModel.pieces→
        select (info | info.__class__.__name__ = 'info').GetCompleteInfo()
FutureDataFlowInfo = futureContextModel.pieces→
        select (info | info.__class__.__name__ = 'info').GetCompleteInfo()
```

The method *GetCompleteInfo()* gives us several types of information about a data flow: the name given to the data flow and the sender and the receivers of this data flow. The following step is to make the disjunction between these two sets of data flows. Once again, this being quite simple, only the use of OCL queries is necessary. Here they are:

```
Check_1_2_Part1 = CurrentDataFlowInfo→
                  select (info | not FutureDataFlowInfo→ includes(info))
Check_1_2_Part2 = FutureDataFlowInfo→
                  select (info | not CurrentDataFlowInfo→ includes(info))
```

All the elements of these two lists are then included in the MS Word table used to present the results of the check (see table 5.2). The table is then included in the MS Word file generated by the QROSS-Checker.

MS Word table

Check 1.2 *Every Data flow in the context model for the current system is a candidate data flow for the context model of the future system.*		
Do you want to remove these data flows to create your new system?		
	Comments from City	Comments from others
Output_1_2_Part1_1		
Output_1_2_Part1_2		
...		
Do you want to add these data flows to create your new system?		
	Comments from City	Comments from others
Output_1_2_Part2_1		
Output_1_2_Part2_2		
...		

Table 5.2: Table containing the results of the check 1.2

5.2.3 Check 1.3

Original statement

> *Every use case in the use case diagram for the current system is a candidate for a use case in the use case diagram of the future system.*

Precise statement

Once again, the use of the word "candidate" makes us translate the previous check by two sentences:

> *For each use case found in the use case diagram for the current system but not found in the use case diagram for the future system, are we sure we want to remove it from the use case diagram for the future system?*
> *For each use case found in the use case diagram for the future system but not found in the use case diagram for the current system, are we sure we want to add it to the use case diagram for the future system?*

Implementation

We follow exactly the same reasoning, i.e. we collect the two lists of use cases, one for each diagram, and then make the disjunction between the two sets of use cases. Here are the OCL requests to collect the use cases:

```
CurrentUseCases = currentUseCaseDiagram.pieces→
                  select (uc | uc.__class__.__name__ = 'UseCase')
                  .GetName2()
FutureUseCases = futureUseCaseDiagram.pieces→
                  select (uc | uc.__class__.__name__ = 'UseCase')
                  .GetName2()
```

We then make the disjunction:

```
Check_1_3_Part1 = CurrentUseCases→
                  select( uc | not FutureUseCases→ includes(uc))
Check_1_3_Part2 = FutureUseCases→
                  select (uc | not CurrentUseCases→ includes(uc))
```

Once again, the elements of these two lists are simply included in the MS word file containing the results of the checks (see table 5.3).

MS Word table

Check 1.3 *Every use case in the use case diagram for the current system is a candidate for a use case in the use case diagram for the future system.*		
Do you want to remove these use cases to create your new system?		
	Comments from City	Comments from others
Output_1_3_Part1_1		
Output_1_3_Part1_2		
...		
Do you want to add these use cases to create your new system?		
	Comments from City	Comments from others
Output_1_3_Part2_1		
Output_1_3_Part2_2		
...		

Table 5.3: Table containing the results of the check 1.3

5.2.4 Check 1.4

Original statement

> *Every adjacent actor which communicates directly with the technical system (level 1) in the context model for the current system is a candidate actor for the use case diagram of the current system.*

Precise statement

> *For each adjacent actor, i.e. which communicates directly with the technical system (level 1) in the context model for the current system, do you need to have it present in the use case diagram of the current system?*
> *For each adjacent actor present in the use case diagram for the current system, does it have to be an adjacent actor, i.e. to communicate directly with the technical system (level 1) in the context model for the current system?*

We have to point out here that the expression "communicates directly" means "there is a data flow between the two actors (with no intermediary)". So, we can give a simple definition to the concept of "adjacent actor":

An actor from a context model is an "adjacent actor" if:

- it is an actor from level 2, 3 or 4

and

- there exists at least one direct data flow between this actor and one of the subsystems within level 1

Implementation

The first OCL query collects the names of the actors of the use case diagram for the current system.

```
CurrentUseCaseActors = currentUseCaseDiagram.pieces→
        select (actor | actor.__class__.__name__ = 'UseCaseActor').GetName2()
```

The following phase consists in collecting the adjacent actors from the current context model. To this end, we need to distinguish the actors from level 1 and actors coming from other levels.
We first collect the actors of the level 1.

```
CurrentLevel1 = currentContextModel.pieces→
        select (actor | actor.__class__.__name__ = 'ContextTechnicalAgent')
```

We use the list obtained to collect the "adjacent" data flows, i.e. the data flows involving only one actor from level 1. For more readability, we used two OCL requests. The first one collects all the data flows. The last one uses the first one to collect, among the data flows, the ones which are "adjacent".

```
CurrentDataFlows = currentContextModel.pieces→
        select(df | df.__class__.__name__ = 'ContextInformation')
CurrentAdjDataFlows = CurrentDataFlows→select(df |
        ((CurrentLevel1→includes(df.GetFirstSender())
           and
        (not CurrentLevel1→includes(df.GetFirstReceiver())))
        or
        (CurrentLevel1→includes(df.GetFirstReceiver())
           and
        (not CurrentLevel1→includes(df.GetFirstSender()))))
```

We then have to determine which actors are actually adjacent actors, i.e. exchange data flows with the level 1. They correspond to actors not coming from level 1, involved in at least one adjacent data flow. So, we use the last OCL request we made, to select the actors we are interested in. From the data flows collected by *CurrentAdjDataFlows* we collect all their senders and their receivers, then removing the senders and receivers coming from level 1. Finally, we make this collection a set to avoid repetitions in the result produced.

```
CurrentAdjActors = CurrentAdjDataFlows.GetAllSenders().GetName2()
        →union(CurrentAdjDataFlows.GetAllReceivers().GetName2())
        →select(actor | not CurrentLevel1→includes(actor))→asSet()
```

The final step consists in computing the disjunction between the adjacent actors from the context model for the current system, collected by *CurrentAdjActors*, and the use case actors for the current system, collected by *CurrentUseCaseActors*. This is done with the two following OCL queries:

```
Check_1_4_Part1 = CurrentAdjActors→
        select(actor | not CurrentUseCaseActors→includes(actor))
Check_1_4_Part2 = CurrentUseCaseActors→
        select(actor | not CurrentAdjActors→includes(actor))
```

Each element of these lists is then simply put in the right place in the text file containing the results of the checks (see table 5.4).

MS Word table

Check 1.4 *Every adjacent actor which communicates directly with the technical system (level 1) in the context model for the current system is a candidate actor for the use case diagram for the current system.*		
These adjacent actors from the context model of the current system are not present in the Use Case diagram of the current system:		
	Comments from City	Comments from others
Output_1_4_Part1_1		
Output_1_4_Part1_2		
...		
These actors from the Use Case diagram of the current system are not present in the Context model of the current system:		
	Comments from City	Comments from others
Output_1_4_Part2_1		
Output_1_4_Part2_2		
...		

Table 5.4: Table containing the results of the check 1.4

Example

We build here a simple example to illustrate what we have just said. Figure 5.1 represents the context model, and figure 5.2 represents the use case diagram, both for our current system.

The first thing to do is to identify the adjacent actors in the context model. Stephen, his girlfriend and Andrew are adjacent actors. But Stephen's mother is not one of those because she does not communicate directly with the system within level 1, i.e. Stephen's and Andrew's computers.

The next step consists in identifying the actors in the use case diagram. We have three actors: Stephen, Andrew and Andrew's girlfriend. The result of the check 1.4 gives us the disjunction between these two sets of actors, as given in the table 5.5.

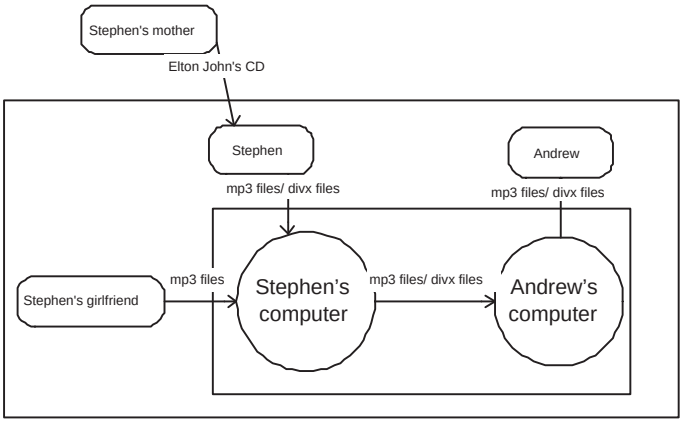

Figure 5.1: Example: context model for the current system

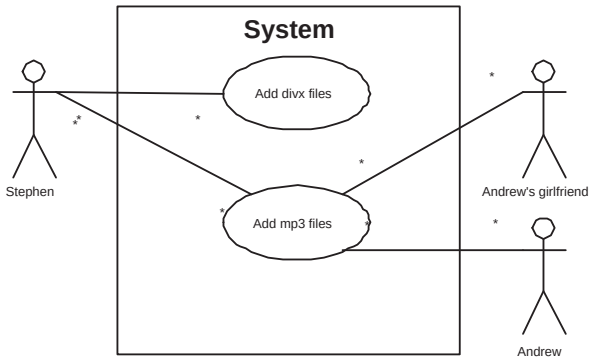

Figure 5.2: Example: use case diagram for the current system

Check 1.4 *Every adjacent actor which communicates directly with the technical system (level 1) in the context model for the current system is a candidate actor for the use case diagram for the current system*		
These adjacent actors from the context model of the current system are not present in the Use Case diagram of the current system:		
	Comments from City	Comments from others
Stephen's girlfriend		
These actors from the Use Case diagram of the current system are not present in the Context model of the current system:		
	Comments from City	Comments from others
Andrew's girlfriend		

Table 5.5: Results of the check 1.4

5.2.5 Check 1.5

Original statement

> *The system boundary in the use case model for the current system should be the same as the boundary between level 1 and 2 in the context model for the current system.*

Difficulties

After discussion with the staff at City University, we found that this check is not really useful. This check actually consists in verifying that the actors involved in the use case diagram are the same as those communicating directly with the level 1 in the context model. This verification is indeed done by applying the check 1.6 (see section 5.2.6).

5.2.6 Check 1.6

Original statement

> *For every data flow from or to level 1 (or one of the sub systems within level 1) of the context model of the current system, there should be a corresponding line in the use case diagram for the current system indicating involvement of the relevant actor in the relevant use case.*

Precise statement

This one was clear enough and did not need to be rewritten. We must only point out that the "or" within "data flow from **or** to level 1" is an exclusive or. Indeed, the possible data flows between subsystems within level 1 are not considered.
This could be translated as:

> *For every data flow involving an adjacent actor and a subsystem from level 1 of the context model of the current system, there should be a corresponding line in the use case diagram for the current system indicating involvement of the relevant actor in the relevant use case.*

Difficulties

The main difficulty to solve here was that there was no precise syntax to name the data flows in the context model. For example, to improve the readability of overloaded context models, the users sometimes use only one arrow to represent several data flows, by writing all the data exchanged in these flows on the arrow. In collaboration with the users, we decided to

separate each data name by a special character. The one we chose is "/". So, it is now possible to determine how much use cases have to be related to an actor in the use case diagram.

Implementation

The data flows from or to level 1 for the context model for the current system have been computed during the check 1.4, under the name "CurrentAdjDataFlows" (see section 5.2.4). We collect the name of the information exchanged, of its senders and of its receivers using the method *GetCompleteInfo()* on *ContextInformation* objects.

```
CurrentAdjDataFlowsInfo = CurrentAdjDataFlows.GetCompleteInfo()
```

We had also to collect information about actors present in the use case diagram and, for each of those actors, the use cases they are involved in. This is done using the following query:

```
CurrentUseCaseInfo = currentUseCaseDiagram.pieces→
        select (actor | actor.__class__.__name__ = 'UseCaseActor')
        .GetCompleteInfo()
```

As we already said, the method *GetCompleteInfo()* on use case actors provides us, for a given actor, with the list of the use cases he is involved in.

We had to apply a treatment on this information because the OCL does not provide us with a function which could allow us to compare strings approximately. Indeed, we need to compare the name of a use case with the name of the information exchanged by a data flow to determine if there is a certain percentage of words being similar. This percentage has been determined empirically to 70. This is done by a function in python, named "StringCompared", which takes two string as arguments and returns the percentage of identical words in those strings.
So, we use python code to check if, for each element of "*CurrentAdjDataFlowsInfo*", we can find a corresponding use case in the list related to the actor from level 2, 3 or 4 involved in the data flow. To identify the actors coming from level 2, 3 and 4, we need to have the list of their names. This is done using a simple OCL request:

```
CurrentLevel234 = currentContextModel.pieces→
        select(actor | actor.type = 'agent' and
        not CurrentLevel1→includes(actor)).GetName2()
```

Let us say that the elements of *CurrentAdjDataFlowsInfo* have the following structure:

- CurrentAdjDataFlowsInfo[0] contains the name of the information exchanged in the data flow.
- CurrentAdjDataFlowsInfo[1] contains the name of the actors sending the information.
- CurrentAdjDataFlowsInfo[2] contains the name of the actors receiving the information.

We use a dictionary to represent values provided by the metric "CurrentUseCaseInfo". The keys of this dictionary are the name of the actors present in the use case diagram, and its values are the lists of use cases in which the actor "key" is involved. We name it "*useCaseActorDict*".
Firstly, we give the pseudo code for this check, and then the python code:

```
actorsLevel234 = Values(CurrentLevel234)
useCaseActors = keys(useCaseActorDict)
missingFlows = []       ##will contain the data flows for which no
                        ##equivalent use case has been found
for each flow in CurrentAdjDataFlowsInfo:
    flowActor234 = ''       ##will contain the sender or receiver coming
                            ##from level 2, 3 or 4.
    if flow.sender ∈ actorsLevel234:
        flowActor234 = flow.sender
    else:
        flowActor234 = flow.receiver
    if flowActor234 ∈ useCaseActors:
        for each info in contextInfo:
            numberOfUseCaseFound = 0
            for each useCase in useCaseActorDict[flowActor234]:
                if Compare(useCase, info) is satisfying:
                    numberOfUseCaseFound = numberOfUseCaseFound+1
            if ((twowayDataFlow and (numberOfUseCaseFound<2))
            or(onewayDataFlow and ((numberOfUseCaseFound=0))):
                missingFlows = missingFlows + flow
    ## if flowActor234 not in use case diagram ##
    else:
        missingFlows = missingFlows + flow
write missingFlows
```

```
bidirectional = 0
actorsLevel234 = project.metricsValues['CurrentLevel234'][0]
for flow in CurrentAdjDataFlowsInfo:
    contextInformations = flow[0]
    senders = flow[1]
    receivers = flow[2]
    flowActor234 = ''  # contains the actor coming from level 2, 3 or 4
                       # involved in the data flow
    if senders[1] != '':
        # bidirectional data flow
        bidirectional = 1
        if senders[0] in level234:
            flowActor234 = senders[0]
        elif senders[1] in level234:
            flowActor234 = senders[1]
    else:
        # unidirectional data flow
        if senders[0] in level234:
            flowActor234 = senders[0]
        elif receivers[0] in level234:
            flowActor234 = receivers[0]
    if ucActorDict.has_key(flowActor234):
        # search for corresponding use case in those of this flowActor234
        contextInformations = contextInformations[0].split('/')
        for info in contextInformations:
            res = 0
            for uc in useCaseActorDict[flowActor234]:
                res2 = self.StringCompared(info.upper(), uc.upper())
                if res2 > 70:
                    res = res+1
            if ((bidirectional==0) and(res<1)):
                write( info+'sent by: '+str(senders[0:-1])+
                    'received by: '+str(receivers[0:-1]))
            elif ((bidirectional==1)and(res<2)):
                write( info+'sent by: '+str(senders[0:])+
                    'received by: ' + str(receivers[0:]))
    else:
        write(contextInformations[0]+'sent by: '+ str(senders[0:-1])+
            'received by: ' + str(receivers[0:-1]) +
            '(cause actor '+flowActor234+'not found in Use Case diagram)')
    bidirectional = 0
```

MS Word table

Check 1.6 *For every data flow from or to level 1 (or one of the sub systems within level 1) of the context model of the current system, there should be a corresponding line in the use case diagram for the current system indicating involvement of the relevant actor in the relevant use case.*		
These data flows of the current context model do not have corresponding use cases in the use case diagram:		
	Comments from City	Comments from others
Output_1_6_1		
Output_1_6_2		
...		

Table 5.6: Table containing the results of the check 1.6

Example

To make the concepts of this check clearer, we illustrate it, using the same schemas as in the example illustrating the check 1.4 (see section 5.2.4), i.e. figure 5.1 for the context model and figure 5.2 for the use case diagram.
We first identify the data flows from or to level 1:

- "mp3 files/divx files" between Stephen and Stephen's computer
- "mp3 files" between Stephen's girlfriend and Stephen's computer
- "mp3 files/divx files" between Andrew and Andrew's computer

The exchange of mp3 and divx files between Stephen's and Andrew's computers is not a data flow we are interested in, since both concerned actors are within level 1. We are not interested either in the flow of Elton John's cd between Stephen's mother and Stephen, since none of the actors is from level 1.
To satisfy the check in the use case diagram, we should have:

- a line between Stephen and a use case involving mp3 files
- a line between Stephen and a use case involving divx files
- a line between Stephen's girlfriend and a use case involving mp3 files
- a line between Andrew and a use case involving mp3 files
- a line between Andrew and a use case involving divx files

But we can see on figure 5.2 that we have:

- a line between Stephen and a use case involving mp3 files
- a line between Stephen and a use case involving divx files
- a line between Andrew's girlfriend and a use case involving mp3 files
- a line between Andrew and a use case involving mp3 files

The results of the check 1.6 are thus given by the table 5.7

Check 1.6 *For every data flow from or to level 1 (or one of the sub system within level 1) in the context model for the current system, there should be a corresponding line in the use case diagram for the current system indicating involvement of the relevant actor in the relevant use case.*		
These data flows of the current context model do not have corresponding use cases in the use case diagram:		
	Comments from City	Comments from others
mp3 files sent by Stephen's girlfriend received by Stephen's computer		
divx files sent by Andrew received by Andrew's computer		

Table 5.7: Results of the check 1.6 for the example.

5.2.7 Check 1.7

Original statement

> *Every adjacent actor (at level 2, 3 or 4) which communicates directly with the technical system (level 1) in the context model for the future system is a candidate actor for the use case diagram for the future system.*

This one is the same as the check 1.4, but applied to context model and use case diagram for the future system. The reasoning is thus similar to the one described in section 5.2.4

5.2.8 Check 1.8

Original statement

> *The system boundary in the use case model for the future system should be the same as the boundary between level 1 and 2 in the context model for the future system.*

This one is the same as the check 1.5 which is included by the check 1.6 (see section 5.2.6), but applied to the schemas for the future system (see section 5.2.5)

5.2.9 Check 1.9

Original statement

> *For every data flow from or to level 1 (or one of the sub systems within level 1) of the context model for the future system, there should be a corresponding line in the use case diagram for the future system indicating involvement of the relevant actor in the relevant use case.*

This check is similar to the check 1.6, applied to context model and use case diagram for the future system (see section 5.2.6).

5.2.10 Check 1.10

Original statement

> *Services and functions related to use cases in the use case diagram for the future system should map to system level requirements, i.e. high level functional and non functional requirements in the requirements database.*

Difficulties

In this check, the main difficulty is the lack of precision in the way that services, functions and functional and non functional requirements are stated. To be effective in the comparison of these, we should use an analyzer for natural language, if we do not want to constrain the users in the definition of these statements. Such issues are out of the scope of this thesis. In dialogue with the members of the RESCUE team, we decided thus to let this check down, because we do not have an analyzer. However, this point will be discuss in chapter 7

5.3 Second phase of synchronisation checks

5.3.1 Check 2.1

Original statement

> *All external actors in the i* SD model of the current system should correspond to actors in the use case descriptions for the current system.*

Precise statement

> *All external actors in the i* SD model of the current system, corresponding to actors from level 3 or 4 in the context model, should correspond to actors in the use case descriptions for the current system.*

Difficulties

One of the issues which we have encountered here is that in i* there was no notational convention to distinguish external and internal actors. This is a knowledge the users of the RESCUE process had, and so did not need to make it explicit. But to make it accessible to QROSS, we needed to transcribe it.
We decided that, instead of marking the external actors, the users should mark the "internal" ones, to facilitate the user work. Indeed, those are used most frequently and the users undoubtedly memorize them more easily. We chose as a convention to mark these by a "*" in their name. The use of a "*" is of course forbidden in names of other actors.
We built a function "isExternal" on objects of type SDActor, which informs us with the fact that an SDActor is external or not.

Implementation

The first thing to do to implement this check is to collect the names of the external actors of the i* SD model. This is done using an OCL request:

```
CurrentExternalSDActors = currentSDModel.pieces→
        select(actor | actor.__class__.__name__ = 'SDActor')→
        select(actor | actor.IsExternal()<>0).GetName2()
```

We also need to have the list of actors enumerated in the different use case descriptions. This is done using the following OCL request.

```
CurrentUseCaseDescrActors = currentUseCaseDiagram.pieces→
        select(useCase | useCase.__class__.__name__ = 'UseCase')
        .GetActors()→asSet()
```

The next stage is to make a disjunction between these two sets using an OCL request:

```
Check_2_1 = CurrentExternalSDActors→
            select(act | not CurrentUseCaseDescrActors→includes(act))
```

The result produced is inserted in the word document produced as the result of running the checker (see table 5.8).

MS Word table

Check 2.1 *All external actors in the i* SD model of the current system should correspond to actors in the use case descriptions for the current system.*		
These external actors in the Strategic dependency model for the current system are not present in the use case descriptions:		
	Comments from City	Comments from others
Output_2_1_1		
Output_2_1_2		
...		

Table 5.8: Table containing the results of the check 2.1 for the example.

Example

To make the concepts of external actors and actors found in the use case descriptions clearer, we build an i* SD model, a use case diagram and a use case description, simplified for the ease of comprehension. Let us say that we have the i* SD model presented on figure 5.3 from [Maid 05a]. We can see that the external actors are “Airline” and “Passenger”, as the two other ones have a “*” within their names.

Let us say that we have the corresponding use case diagram, presented on figure 5.4, and a part of a use case description, found in table 5.9. We can notice that the writer of this use case diagram forgot that “Airline” played a part in this system and did not mention it in the use case description. So, the result produced by running the QROSS-Checker is given in table 5.10.

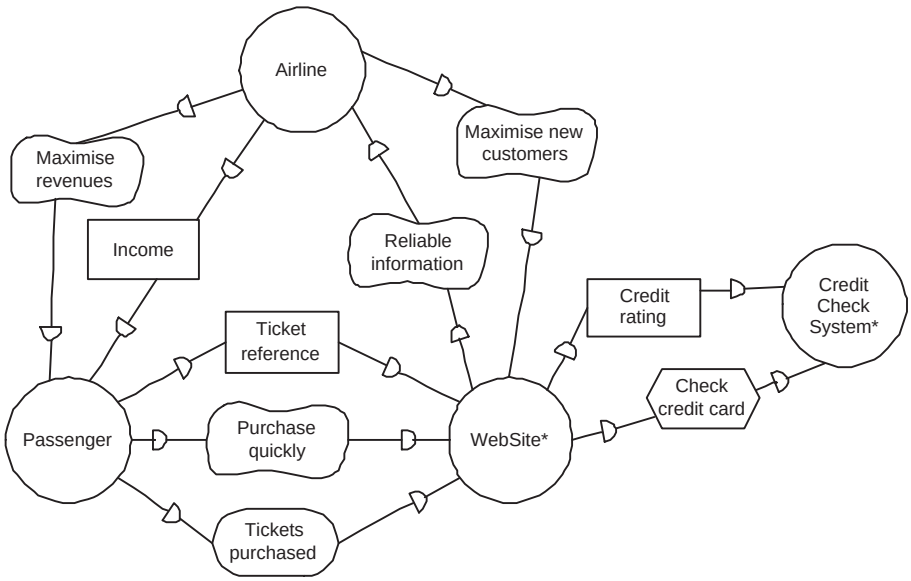

Figure 5.3: i* SD model for the current system.

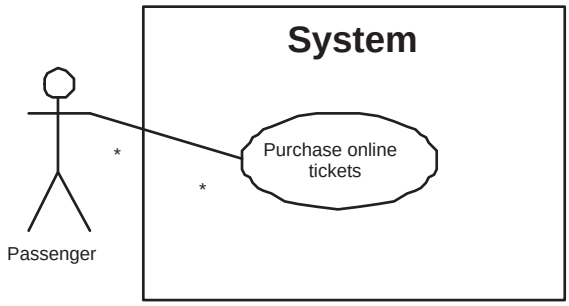

Figure 5.4: Use case diagram for the current system.

Name of use case	Purchase online tickets
Use Case ID	UC1
Author	Me
Date	17/03/2005
Source	A simple mind
Actors	Passenger
Problem statements	None
Precis	The passenger connects himself to the website and buys tickets.
...	...

Table 5.9: Use case description for the current system.

Check 2.1 *All external actors in the i* SD model for the current system should correspond to actors in the use case descriptions for the current system.*		
These external actors in the SD model for the current system are not present in the use case descriptions		
	Comments from City	Comments from others
Airline		

Table 5.10: Results of the check 2.1

5.3.2 Check 2.2

Original statement

> *All external actors in the i* SD model of the current system are candidate actors for the i* SD model for the future system.*

Precise statement

> *For each external actor found in the i* SD model for the current system but not found among all the actors in the i* SD model for the future system, are we sure we want to remove it from the i* SD model for the future system?*
> *For each actor found in the i* SD model for the future system but not found among external actors in the i* SD model for the current system, are we sure we want to add it to the i* SD model for the future system?*

Difficulties

Here again, we faced difficulties we already encountered. They are the use of the word "candidate", as in section 5.2.1, and the problem of determining external actors, as in section 5.3.1. These issues have already been solved.

Implementation

We already have the list of external actors for the i* SD model for the current system, since we collected it in the check 2.1 (see section 5.3.1). So, the first thing we do is collecting all the actors from the i* SD model for the future system, using the following OCL query:

```
FutureSDActors = futureSDModel.pieces→
        select(actor | actor.__class__.__name__ = 'SDActor')
        .GetName2()
```

We then make the disjunction between the two sets of actors.

```
Check_2_2_Part1 = CurrentExternalSDActors→
                  select(act | not FutureSDActors→includes(act))
Check_2_2_Part2 = FutureSDActors→
                  select(act | not CurrentExternalSDActors→includes(act))
```

Finally, we put the element of the obtained lists in the word document containing the results of the checks (see table 5.11).

MS Word table

Check 2.2 *All external actors in the i* SD model of the current system are candidate actors for the i* SD model of the future system.*		
Do you want to remove these external actors from the SD model to create the new system?		
	Comments from City	Comments from others
Output_2_2_Part1_1		
Output_2_2_Part1_2		
...		
Do you want to add these external actors to create the new system?		
	Comments from City	Comments from others
Output_2_2_Part2_1		
Output_2_2_Part2_2		
...		

Table 5.11: Table containing the results of the check 2.2

5.3.3 Check 2.3

Original statement

> *All dependencies in the i* SD model of the current system are candidate dependencies for the i* SD model for the future system.*

Precise statement

> *For each dependency found in the i* SD model for the current system but not found in the i* SD model for the future system, are we sure we want to remove it from the i* SD model for the future system?*
> *For each dependency found in the i* SD model for the future system but not found in the i* SD model for the current system,*

are we sure we want to add it to the i SD model for the future system?*

Difficulties

There were no difficulty, except, as usual, the use of the word "candidate".

Implementation

The implementation is quite simple and only requires the following OCL requests.
The two first ones collect the dependencies found in the i* SD model for the current and the future system:

```
CurrentSDDependency = currentSDModel.pieces→
        select(dep | dep.__class__.__name__ = 'SDResourceDependency'
                or dep.__class__.__name__ = 'SDTaskDependency'
                or dep.__class__.__name__ = 'SDGoalDependency'
                or dep.__class__.__name__ = 'SDSoftGoalDependency')
                .GetCompleteInfo()
FutureSDDependency = futureSDModel.pieces→
        select(dep | dep.__class__.__name__ = 'SDResourceDependency'
                or dep.__class__.__name__ = 'SDTaskDependency'
                or dep.__class__.__name__ = 'SDGoalDependency'
                or dep.__class__.__name__ = 'SDSoftGoalDependency')
                .GetCompleteInfo()
```

The method "*GetCompleteInfo()*" on SD dependencies provides us with all the information we need about these dependencies, i.e. the elements of CurrentSDDependency have the following structure:

- CurrentSDDependency[0] indicates the type of dependency which we deal with, i.e. its value can be "resource", "task", "goal" or "softGoal";
- CurrentSDDependency[1] contains the name of the dependency;
- CurrentSDDependency[2] indicates the depender;
- CurrentSDDependency[3] indicates the dependees.

The two next requests are used to do the disjunction between the two sets of dependencies:

```
Check_2_3_Part1 = CurrentSDDependency→
                  select(dep | not FutureSDDependency→
                  includes(dep))
Check_2_3_Part2 = FutureSDDependency→
                  select(dep | not CurrentSDDependency→
                  includes(dep))
```

MS Word table

Check 2.3 *All dependencies in the i* SD model of the current system are candidate dependencies for the i* SD model of the future system.*		
Do you want to remove these dependencies to create the new system?		
	Comments from City	Comments from others
Output_2_3_Part1_1		
Output_2_3_Part1_2		
...		
Do you want to add these dependencies to create the new system?		
	Comments from City	Comments from others
Output_2_3_Part2_1		
Output_2_3_Part2_2		
...		

Table 5.12: Table containing the results of the check 2.3

5.3.4 Check 2.4

Original statement

> *All actions in the use case descriptions for the current system are candidate actions for the use case descriptions for the future system.*

Precise statement

> *For each action found in the use case descriptions for the current system but not found in the use case descriptions for the future system, are we sure we want to remove it from the use case descriptions for the future system?*
> *For each action found in the use case descriptions for the future system but not found in the use case descriptions for the current system, are we sure we want to add it to the use case descriptions for the future system?*

Difficulties

The main difficulty here is that we have to compare two actions to determine if they are similar. Once again, since we do not have a lexical analyzer for natural language, the only way to do it is to constrain the user. We thus require the user to use the same sentence to describe the same action, in the

current and the future model. This is an acceptable compromise, since it does not charge more the user, and, moreover, it facilitates the traceability.

Implementation

We start by collecting all the actions for use case descriptions for the current and the future models, using OCL requests:

```
CurrentUseCaseActions = currentUseCaseDiagram.pieces→
        select(uc | uc.__class__.__name__ = 'UseCase').GetActions()
FutureUseCaseActions = futureUseCaseTable.pieces→
        select(uc | uc.__class__.__name__ = 'UseCase').GetActions()
```

The next step consists in comparing each element of the list "CurrentUseCaseActions" with each element of the list "FutureUseCaseActions" to determine if the first one have an equivalent in the second list and vice versa. We compare the strings representing these use case actions, using the function "StringCompared" (see section 5.2.6), which returns the percentage of identical words in the two strings. To consider two use case actions as equivalent, this percentage has to be at least 65. It has been determined empirically.
Here is the python code to implement this:

```
cActions= project.metricsValues['CurrentUseCaseActions'][0]
## contains the results of the query "CurrentUseCaseActions ##
fActions = project.metricsValues['FutureUseCaseActions'][0]
## contains the results of the query "FutureUseCaseActions ##
cActionsNonF=[]    #actions found in current use case descriptions
                   # but not in future use case descriptions
for cA in cActions:      #for each action in the current UCD
    res = 0
    for fA in fActions: #for each action in the future UCD
        r = self.StringCompared(cA.upper(), fA.upper())
        if r>res:
            res = r
    if res>65:           # if the actions are equivalent
        pass
    else:                # if the actions are not equivalent
        cActionsNonF.append(cA)
fActionsNonC=[]    #actions found in future use case descriptions
                   # but not in current use case descriptions
for fA in fActions:      #for each action in the future UCD
    res = 0
    for cA in cActions: #for each action in the current UCD
        r = self.StringCompared(fA.upper(), cA.upper())
        if r>res:
            res=r
    if res>65:           # if the actions are equivalent
        pass
    else:                # if the actions are not equivalent
        fActionsNonC.append(fA)
```

We then include the elements of the lists *cActionsNonF* and *fActionsNonC* in the word document generated by running the QROSS-Checker (see table 5.13).

MS Word table

Check 2.4 *All actions in use case descriptions for the current system are candidate actions for use case descriptions of the future system.*		
Do you want to remove these actions to create the new system?		
	Comments from City	Comments from others
Output_2_4_Part1_1		
Output_2_4_Part1_2		
...		
Do you want to add these actions to create the new system?		
	Comments from City	Comments from others
Output_2_4_Part2_1		
Output_2_4_Part2_2		
...		

Table 5.13: Table containing the results of the check 2.4

5.3.5 Check 2.5

Original statement

> *All variations in the use case descriptions for the current system are candidate variations or use cases for the future system.*

Precise statement

> *For each variation found in the use case descriptions for the current system but not found in the use case descriptions for the future system, are we sure we want to remove it from the use case descriptions for the future system?*
> *For each variation found in the use case descriptions for the future system but not found in the use case descriptions for the current system, are we sure we want to add it to the use case descriptions for the future system?*

This check being similar to the check 2.4, but applied to the variations instead of actions, the reasoning is similar as well.

5.3.6 Check 2.6

Original statement

> *All external actors in the i* SD model of the future system should correspond to actors in the use case descriptions for the future system.*

This check being similar to the check 2.1, applied to the future system, the reasoning is similar (see section 5.3.1 for more details).

5.3.7 Check 2.7

Original statement

> *For all task dependencies identified in the i* SD model of the future system, which represent tasks carried out by actors in the use case diagram, there should be a part of a use case description which describes how those tasks are carried out.*

Difficulties

Not having a natural language analyzer, it seemed impossible to relate the name of a task dependency with a description. The only thing we could do was to determine if, in a use case description, we could find the actor doing the task involved in the task dependency, and then searching in this use case description if we could find the name of the task dependency. That proved not to be sufficient.
We decided thus in dialogue with the City team to let this check down for the QROSS-Checker. However, this check need to be done manually. We proposed thus to provide the list of task dependencies for the i* SD model, for the ease of the users. Collecting these dependencies could indeed be hard, and there is always a danger to forget one or several of these. This list is provided in the word document containing the results of the checks.

Implementation

Here is the OCL request used to collect the task dependencies for the i* SD model for the future system:

```
FutTaskDependencies = futureSDModel.pieces→
        select(dep | dep.__class__.__name__ = 'SDTaskDependency')
        .GetCompleteInfo()
```

MS Word table

Check 2.7 *For all task dependencies identified in the i* SD model of the future system, which represent tasks carried out by actors in the use case diagram, there should be a part of a use case description which describes how those tasks are carried out.*		
Here is a list of the task dependencies in the SDModel for the future system:		
	Comments from City	Comments from others
FutTaskDeps_1		
FutTaskDeps_2		
...		

Table 5.14: Table containing the results of the check 2.7

5.3.8 Check 2.8

Original statement

> *All requirements associated with use cases using the RESCUE use case template should be stored in the requirements database.*

Precise statement

> *All requirements we find in the word documents corresponding to use case descriptions made with the RESCUE use case template should be stored in the requirements database.*

Difficulties

The main difficulty encountered here is the lack of time. Being the last check to be implemented, this one has not been finished. We needed to clarify the source of the word documents containing the use case description. There are indeed two sources where we can find these word documents. The use case descriptions are recopied manually within RequisitePro to create the requirements database.
Moreover, we did not know if the name given to the requirements were the same in the word documents and in the database.
For all these reasons, this check was not developed at this stage.

Chapter 6

Application to the case study

We present here the results of applying our QROSS-Checker to the case study introduced in chapter 3. For each check, we provide the MS Word table produced as well as a small explanation.
Note that this is a post-mortem application of our tool tot he case study. The case was used for testing purposes and, here, for illustration but not for actually checking the models and documents during the project development.

6.1 First phase of synchronisation checks

6.1.1 Check 1.1

This check compares the actors coming from the context model for the current system (see figure 3.1, page 21) with the actors coming from the context model for the future system (see figure 3.3, page 22). If we examine these two figures, we can see that two actors are present in the model for the current system but not in the model for the future system: Indicator and Road-side beacon. The same way, we can see that two actors are present in the model for the future system but not in the model for the current system: GPS and Display. Table 6.1 shows that the results obtained for the check 1.1 by applying the QROSS-Checker to the context models for the Countdown system are those we expected.

Check 1.1 *Every actor in the context model for the current system is a candidate actor for the context model of the future system.*		
Do you want to remove these actors to create your new system?		
	Comments from City	Comments from others
INDICATOR		
ROAD-SIDE BEACON		
Do you want to add these actors to create your new system?		
	Comments from City	Comments from others
GPS		
DISPLAY		

Table 6.1: Results of the check 1.1 for the Countdown system.

6.1.2 Check 1.2

Similarly to the check 1.1, this check compares the data flows from the context model for the current system (see figure 3.1, page 21) with the data flows from the context model for the future system (see figure 3.3, page 22). Examining these two figures, we notice that the following data flows stand in the model for the current system but not in the model for the future system:

- Bus locations from On-board bus system to AVL system;
- Bus information / route information / traffic information between AVL system and Route controller;
- Bus information from AVL system to Indicator;
- Bus information from Indicator to Passenger;
- Beacon ID from Road-side beacon to On-board bus system.

The same way, we can see that the following data flows stand in the model for the future system but not in the model for the current system:

- Bus information / route information / traffic information from AVL system to Route controller;
- Bus information from AVL system to Display;
- Bus information from Display to Passenger;
- Bus signal from On-board bus system to GPS.

Table 6.2 shows that the results obtained for the check 1.2 by applying the QROSS-Checker to the context models for the Countdown system are those we expected.

Check 1.2 *Every data flow in the context model for the current system is a candidate data flow for the context model of the future system.*		
Do you want to remove these data flows to create your new system?		
	Comments from City	Comments from others
BUS LOCATIONS sent by: ['ON-BOARD BUS SYSTEM'] received by: ['AVL SYSTEM']		
BUS INFORMATION/ ROUTE INFORMATION/ TRAFFIC INFORMATION sent by: ['AVL SYSTEM', 'ROUTE CONTROLLER'] received by: ['ROUTE CONTROLLER', 'AVL SYSTEM']		
BUS INFORMATION sent by: ['AVL SYSTEM'] received by: ['INDICATOR']		
BUS INFORMATION sent by: ['INDICATOR'] received by: ['PASSENGER']		
BEACON ID sent by: ['ROAD-SIDE BEACON'] received by: ['ON-BOARD BUS SYSTEM']		
Do you want to add these data flows to create your new system?		
	Comments from City	Comments from others
BUS INFORMATION/ ROUTE INFORMATION/ TRAFFIC INFORMATION sent by: ['AVL SYSTEM'] received by: ['ROUTE CONTROLLER']		
BUS INFORMATION sent by: ['AVL SYSTEM'] received by: ['DISPLAY']		
BUS INFORMATION sent by: ['DISPLAY'] received by: ['PASSENGER']		
BUS LOCATIONS sent by: ['GPS'] received by: ['AVL SYSTEM']		
BUS SIGNAL sent by: ['ON-BOARD BUS SYSTEM'] received by: ['GPS']		

Table 6.2: Results of the check 1.2 for the Countdown system.

6.1.3 Check 1.3

This check compares the use cases found in the use case diagram for the current system (see figure 3.2 on page 22) with those found in the use case diagram for the future system (see figure 3.4 on page 23). By looking at them, we can see that all the use cases present in the use case diagram for the current system are found in the use case diagram for the future system and vice versa. The table presenting the results obtained by running the QROSS-Checker on these diagrams will thus be empty.

Check 1.3 *Every use case in the use case diagram for the current system is a candidate for a use case in the use case diagram for the future system.*		
Do you want to remove these use cases to create your new system?		
	Comments from City	Comments from others
Do you want to add these use cases to create your new system?		
	Comments from City	Comments from others

Table 6.3: Results of the check 1.3 for the Countdown system.

6.1.4 Check 1.4

This check compares the adjacent actor from the context model for the current system (see figure 3.1 on page 21) with the actors found in the use case diagram for the current system (see figure 3.2 on page 22). The adjacent actors are enumerated below:

- Road-side beacon;
- Indicator;
- Driver;
- Route controller;
- London transport.

By observing the use case diagram, we can see that the list above corresponds to the actors found on the use case diagram. Thus, the table presenting the results of this check will be empty.

Check 1.4 *Every adjacent actor which communicates directly with the technical system (level 1) in the context model for the current system is a candidate actor for the use case diagram for the current system.*		
These adjacent actors from the context model of the current system are not present in the Use Case diagram of the current system:		
	Comments from City	Comments from others
These actors from the Use Case diagram of the current system are not present in the Context model of the current system:		
	Comments from City	Comments from others

Table 6.4: Results of the check 1.4 for the Countdown system.

6.1.5 Check 1.5

Although this check is not implemented, we chose to put its statement in the word document produced by the QROSS-Checker, to prevent the users being disturbed by the fact that a check is missing. This way, we also respect the original numbering of the checks.

Check 1.5 *The system boundary in the use case model for the current system model should be the same as the boundary between levels 1 & 2 in the context model for the current system.*

Table 6.5: Statement of the check 1.5 (not implemented).

6.1.6 Check 1.6

Here, we check that for every data flow from or to level 1 in the context model for the current system (see figure 3.1 on page 21), there is a corresponding line in the use case diagram for the current system (see figure 3.2 on page 22) indicating involvement of the relevant actor in the relevant use case. Here is the list of data flows from or to level 1 for the context model for the current system:

- Beacon ID from Road-side beacon to On-board bus system;
- Bus information from AVL system to Indicator;
- Bus information / route information / traffic information between AVL system and Route controller;
- Bus route from Driver to On-board bus system;
- Traffic information from AVL System to London Transport.

We give the list of the use cases found in the use case diagram for the current system, related to the actor involved in:

- Driver: Alert controller to emergency;
- Driver: Operator sign-on;
- Indicator: Provide information for travel decisions;
- Road-side beacon: Determine bus location and arrival time;
- Route controller: Send route information;
- Route controller: Monitor bus location;
- London Transport: Receive current traffic information.

The matches found by the QROSS-Checker are the correspondences between the data flow “Traffic information from AVL System to London Transport” and the use case “receive current traffic information“ related to the actor “London Transport”, and between the data flow “route information between AVL System and Route controller” and the use case “send route information” related to the actor ‘Route controller”. All the other data flows will thus appear in the table presenting the results of this check.

Check 1.6
For every data flow from or to level 1 (or one of the sub systems within level 1) of the context model of the current system, there should be a corresponding line in the use case diagram for the current system indicating involvement of the relevant actor in the relevant use case.

These data flows of the current context model do not have corresponding use cases in the use case diagram:

	Comments from City	Comments from others
BUS INFORMATION sent by: ['AVL SYSTEM', 'ROUTE CONTROLLER'] received by: ['ROUTE CONTROLLER', 'AVL SYSTEM'] (2 use cases missing (1 for each direction))		
ROUTE INFORMATION sent by: ['AVL SYSTEM', 'ROUTE CONTROLLER'] received by: ['ROUTE CONTROLLER', 'AVL SYSTEM'] (1 use case missing (for 1 of the directions))		
TRAFFIC INFORMATION sent by: ['AVL SYSTEM', 'ROUTE CONTROLLER'] received by: ['ROUTE CONTROLLER', 'AVL SYSTEM'] (2 use cases missing (1 for each direction))		
BUS ROUTE sent by: ['DRIVER'] received by: ['ON-BOARD BUS SYSTEM']		
BUS INFORMATION sent by: ['AVL SYSTEM'] received by: ['INDICATOR']		
BEACON ID sent by: ['ROAD-SIDE BEACON'] received by: ['ON-BOARD BUS SYSTEM']		

Table 6.6: Results of the check 1.6 for the Countdown system.

6.1.7 Check 1.7

This check is similar to the check 1.4 and compares the adjacent actor from the context model for the future system (see figure 3.3 on page 22) with the actors found in the use case diagram for the future system (see figure 3.4 on page 23). The adjacent actors are enumerated below:

- GPS;
- Display;
- Driver;
- Route controller;
- London transport.

By observing the use case diagram, we can see that the list above corresponds to the actors found on the use case diagram. Thus, like for the check 1.4, the table presenting the results of this check is empty.

Check 1.7 *Every adjacent actor (level 2, 3 or 4) which communicates directly with the technical system (level 1) in the context model for the future system is a candidate actor for the use case diagram for the future system.*		
These adjacent actors from the context model of the future system are not present in the Use Case diagram of the future system:		
	Comments from City	Comments from others
These actors from the Use Case diagram of the future system are not present in the Context model of the future system:		
	Comments from City	Comments from others

Table 6.7: Results of the check 1.7 for the Countdown system.

6.1.8 Check 1.8

For the same reason as for the check 1.5, we indicate the statement of the check in the word document produced by running the QROSS-Checker.

Check 1.8 *The system boundary in the use case model for the future system model should be the same as the boundary between levels 1 & 2 in the context model for the future system.*

Table 6.8: Statement of the check 1.8 (not implemented).

6.1.9 Check 1.9

Similarly as for check 1.6, we check that for every data flow from or to level 1 in the context model for the future system (see figure 3.3 on page 22), there is a corresponding line in the use case diagram for the future system (see figure 3.4 on page 23) indicating involvement of the relevant actor in the relevant use case. Here is the list of data flows from or to level 1 for the context model for the future system:

- Bus locations from GPS to On-board bus system;
- Bus signal from On-board bus system to GPS;
- Bus information from AVL system to Display;
- Bus route from Driver to On-board bus system;
- Bus information / route information / traffic information from AVL system to Route controller;
- Traffic information from AVL System to London Transport.

We give the list of the use cases found in the use case diagram for the future system, related to the actor involved in:

- Driver: Alert controller to emergency;
- Driver: Operator sign-on;
- Display: Provide information for travel decisions;
- GPS: Determine bus location and arrival time;
- Route controller: Send route information;
- Route controller: Monitor bus location;
- London Transport: Receive current traffic information.

The matches found by the QROSS-Checker are the correspondences between the data flow “Traffic information from AVL System to London Transport” and the use case “receive current traffic information“ related to the actor “London Transport” and between the data flow “route information between AVL System and Route controller” and the use case “send route information” related to the actor ‘Route controller”. All the other data flows will thus appear in the table presenting the results of this check.

Check 1.9 *For every data flow from or to level 1 (or one of the sub systems within level 1) of the context model of the future system, there should be a corresponding line in the use case diagram for the future system indicating involvement of the relevant actor in the relevant use case.*		
These data flows of the future context model do not have corresponding use cases in the use case diagram:		
	Comments from City	Comments from others
BUS INFORMATION sent by: ['AVL SYSTEM'] received by: ['ROUTE CONTROLLER']		
TRAFFIC INFORMATION sent by: ['AVL SYSTEM'] received by: ['ROUTE CONTROLLER']		
BUS INFORMATION sent by: ['AVL SYSTEM'] received by: ['DISPLAY']		
BUS ROUTE sent by: ['DRIVER'] received by: ['ON-BOARD BUS SYSTEM']		
BUS SIGNAL sent by: ['ON-BOARD BUS SYSTEM'] received by: ['GPS']		

Table 6.9: Results of the check 1.9 for the Countdown system.

6.1.10 Check 1.10

Once again, for the same reason as for check 1.5, we give the statement of the check in the MS Word document presenting the results of the checks obtained by running the QROSS-Checker.

Check 1.10 *Services and functions related to use cases in the use case diagram for the future system should map to system level requirements, i.e. high level functional and non functional requirements in the requirements database.*

Table 6.10: Statement of the check 1.10 (not implemented).

6.2 Second phase of synchronisation checks

6.2.1 Check 2.1

This check compares the external actors from the i* SD model for the current system (see figure 3.5 on page 24) with the actors from use case descriptions for the current system (see tables 3.1 and 3.2 on pages 25 and 26). The external actors (i.e. those not marked by a "*") in the SD model for the current system are:

- Comms System;
- Passenger;
- Road-side beacon;
- London Transport.

The actors found in the only description we have are:

- Indicator;
- Passenger;
- AVL System.

We can thus conclude that the results provided in the table 6.11 obtained by running the QROSS-Checker on documents produced for the Countdown system are those expected.

Check 2.1 *All external actors in the i* SD model of the current system should correspond to actors in the use case descriptions for the current system.*		
These external actors in the Strategic dependency model for the current system are not present in the use case descriptions:		
	Comments from City	Comments from others
ROAD-SIDE BEACON		
LONDON TRANSPORT		
COMMS SYSTEM		

Table 6.11: Results of the check 2.1 for the Countdown system.

6.2.2 Check 2.2

Here, we check whether each external actor in the current SD model (see figure 3.5 on page 24) is present in some shape or form in the SD model for the future system (see figure 3.6 on page 27), i.e. we check no actors we know about in the current system had been forgotten about in the model of the future system. The list of external actors in the SD model for the current system is given in section 6.2.1. The actors in the SD model are:

- Comms System;
- Driver;
- Passenger;
- GPS;
- Route Controller;
- On-board Bus System;
- London Transport;
- AVL System;
- Countdown Display.

Thus, we should find in the table presenting the result that the external actor "Road-Side Beacon" from the SD model for the current system is not represented among the actors of the SD model for the future system, and that the actors Driver, GPS, Route Controller, On-board Bus System, AVL System and Countdown Display are not represented among external actors in the SD model for the current system. We can thus conclude that the results provided by the QROSS-Checker shown in table 6.12 are those we expected.

Check 2.2 *All external actors in the i* SD model of the current system are candidate actors for the i* SD model of the future system.*		
Do you want to remove these external actors from the SD model to create the new system?		
	Comments from City	Comments from others
ROAD-SIDE BEACON		
Do you want to add these actors to create the new system?		
	Comments from City	Comments from others
DRIVER		
AVL SYSTEM		
ROUTE CONTROLLER		
COUNTDOWN DISPLAY		
ON-BOARD BUS SYSTEM		
GPS		

Table 6.12: Results of the check 2.2 for the Countdown system.

6.2.3 Check 2.3

This check compares the dependencies from the i* SD model for the current system (see figure 3.5 on page 24) with the dependencies from the i* SD model for the future system (see figure 3.6 on page 27). From the i* SD model for the current system, we collect the following dependencies:

C1. Driver depends on Comms System for the goal "emergency reported";

C2. Driver depends on Comms System for the soft-goal "communication doesn't cause danger";

C3. Driver depends on Route Controller for the goal "route changes be received";

C4. Driver depends on Route Controller for the soft-goal "response received quickly";

C5. Driver depends on On-Board Bus System for the soft-goal "be easy to use";

C6. Driver depends on On-Board Bus System for the soft-goal "operate without error";

C7. Driver depends on On-Board Bus System for the goal "destination and user ID be input";

C8. Driver depends on On-Board Bus System for the resource "time card";

C9. Passenger depends on Driver for the goal "get to destination";

C10. Passenger depends on Countdown Indicator for the soft-goal "indicator be readable";

C11. Passenger depends on Countdown Indicator for the soft-goal "information be reliable";

C12. Passenger depends on Countdown Indicator for the soft-goal "information be accurate";

C13. Passenger depends on Countdown Indicator for the task "make travel decisions";

C14. Passenger depends on Countdown Indicator for the resource "bus arrival information";

C15. Countdown Indicator depends on AVL System for the soft-goal "information be reliable";

C16. Countdown Indicator depends on AVL System for the resource "bus information";

C17. Countdown Indicator depends on AVL System for the task "display bus information";

C18. Countdown Indicator depends on AVL System for the soft-goal "information be up-to-date";

C19. AVL System depends on On-Board Bus System for the task “calculate arrival times”;

C20. AVL System depends on On-Board Bus System for the soft-goal “bus locations be up-to-date”;

C21. AVL System depends on On-Board Bus System for the goal “buses located”;

C22. On-Board Bus System depends on Road-Side Beacon for the goal “buses located”;

C23. London Transport depends on Route Controller for the goal “bus information monitored”;

C24. London Transport depends on Route Controller for the goal “traffic be updated”;

C25. Route Controller depends on Comms System for the goal “driver contacted”;

C26. Route Controller depends on Comms System for the soft-goal “system be available”;

C27. Route Controller depends on Comms System for the soft-goal “communication be understandable”;

C28. Route Controller depends on AVL System for the goal “bus location monitored”;

C29. Route Controller depends on AVL System for the soft-goal “information be accurate”;

C30. Route Controller depends on AVL System for the goal “route changes decided”;

C31. Route Controller depends on AVL System for the soft-goal “manage routes effectively”;

C32. Route Controller depends on AVL System for the goal “current bus information received”;

C33. Route Controller depends on AVL System for the soft-goal “information be reliable”.

From the i* SD model for the future system, we collect these dependencies:

F1. Driver depends on Comms System for the goal “emergency reported”;

F2. Driver depends on Comms System for the soft-goal “communication doesn’t cause danger”;

F3. Driver depends on Route Controller for the goal “route changes be received”;

F4. Driver depends on Route Controller for the soft-goal “response received quickly”;

F5. Driver depends on On-Board Bus System for the soft-goal “be easy to use”;

F6. Driver depends on On-Board Bus System for the soft-goal “operate without error”;

F7. Driver depends on On-Board Bus System for the goal “destination and user ID be input”;

F8. Driver depends on On-Board Bus System for the resource “time card”;

F9. Passenger depends on Driver for the goal “get to destination”;

F10. Passenger depends on Countdown Display for the soft-goal “display be readable”;

F11. Passenger depends on Countdown Display for the soft-goal “information be reliable”;

F12. Passenger depends on Countdown Display for the soft-goal “information be accurate”;

F13. Passenger depends on Countdown Display for the task “make travel decisions”;

F14. Passenger depends on Countdown Display for the resource “bus arrival information”;

F15. Countdown Display depends on AVL System for the soft-goal “information be reliable”;

F16. Countdown Display depends on AVL System for the resource “bus information”;

F17. Countdown Display depends on AVL System for the task “display bus information”;

F18. Countdown Display depends on AVL System for the soft-goal “information be up-to-date”;

F19. AVL System depends on On-Board Bus System for the task “calculate arrival times”;

F20. AVL System depends on On-Board Bus System for the soft-goal “bus locations be up-to-date”;

F21. AVL System depends on On-Board Bus System for the goal “buses located”;

F22. On-Board Bus System depends on GPS for the goal “buses located”;

F23. GPS depends on On-Board Bus System for the resource “GPS receiver”;

F24. London Transport depends on Route Controller for the goal “bus information monitored”;

F25. London Transport depends on Route Controller for the goal “traffic be updated”;

F26. Route Controller depends on Comms System for the goal “driver contacted”;

F27. Route Controller depends on Comms System for the soft-goal “system be available”;

F28. Route Controller depends on Comms System for the soft-goal “communication be understandable”;

F29. Route Controller depends on AVL System for the goal “bus location monitored”;

F30. Route Controller depends on AVL System for the soft-goal “information be accurate”;

F31. Route Controller depends on AVL System for the goal “route changes decided”;

F32. Route Controller depends on AVL System for the soft-goal “manage routes effectively”;

F33. Route Controller depends on AVL System for the goal “current bus information received”;

F34. Route Controller depends on AVL System for the soft-goal “information be reliable”.

The dependencies 1–9 and 19–21 are the same in both lists and do not appear in the table containing the results of the check, as well as dependencies C23–C33 in the first list and F24–F34 in the second list.
Dependencies 10–18 and 22 are similar in both lists, except that the name of the actors “Countdown Indicator” and “Road-Side Beacon” in the first list become “Countdown Display” and “GPS” respectively in the second list. These dependencies appear thus in the tables 6.13 and 6.14 presenting the results.
The dependency F23 in the second list does not have a correspondence in the first one and so appears in the table 6.14 presenting the results.

Check 2.3 *All dependencies in the i* SD model of the current system are candidate dependencies for the i* SD model of the future system.*		
Do you want to remove these dependencies to create the new system?		
	Comments from City	Comments from others
Goal dependency 'BUS LOCATED' - depender: ON-BOARD BUS SYSTEM - dependees: ROAD-SIDE BEACON		
Resource dependency 'BUS INFORMATION - depender: COUNTDOWN INDICATOR - dependees: AVL SYSTEM*		
SoftGoal dependency 'INFORMATION BE RELIABLE' - depender: COUNTDOWN INDICATOR - dependees: AVL SYSTEM*		
SoftGoal dependency 'INFORMATION BE UP-TO-DATE' - depender: COUNTDOWN INDICATOR - dependees: AVL SYSTEM*		
Resource dependency 'BUS ARRIVAL INFORMATION' - depender: PASSENGER - dependees: COUNTDOWN INDICATOR		
SoftGoal dependency 'INFORMATION BE RELIABLE' - depender: PASSENGER - dependees: COUNTDOWN INDICATOR		
SoftGoal dependency 'INFORMATION BE ACCURATE' - depender: PASSENGER - dependees: COUNTDOWN INDICATOR		
Task dependency 'MAKE TRAVEL DECISIONS' - depender: PASSENGER - dependees: COUNTDOWN INDICATOR		
Task dependency 'DISPLAY BUS INFORMATION' - depender: COUNTDOWN INDICATOR - dependees: AVL SYSTEM*		
SoftGoal dependency 'INDICATOR BE READABLE' - depender: PASSENGER - dependees: COUNTDOWN INDICATOR		

Table 6.13: Results of the check 2.3 for the Countdown system (Part 1).

Do you want to add these dependencies to create the new system?		
	Comments from City	Comments from others
Resource dependency 'GPS RECEIVER' - depender: GPS - dependees: ON-BOARD BUS SYSTEM		
Goal dependency 'BUS LOCATED' - depender: ON-BOARD BUS SYSTEM - dependees: GPS		
Resource dependency 'BUS INFORMATION' - depender: COUNTDOWN DISPLAY - dependees: AVL SYSTEM*		
SoftGoal dependency 'INFORMATION BE RELIABLE' - depender: COUNTDOWN DISPLAY - dependees: AVL SYSTEM*		
SoftGoal dependency 'INFORMATION BE UP-TO-DATE' - depender: COUNTDOWN DISPLAY - dependees: AVL SYSTEM*		
Resource dependency 'BUS ARRIVAL INFORMATION' - depender: PASSENGER - dependees: COUNTDOWN DISPLAY		
SoftGoal dependency 'INFORMATION BE RELIABLE' - depender: PASSENGER - dependees: COUNTDOWN DISPLAY		
SoftGoal dependency 'INFORMATION BE ACCURATE' - depender: PASSENGER - dependees: COUNTDOWN DISPLAY		
Task dependency 'MAKE TRAVEL DECISIONS' - depender: PASSENGER - dependees: COUNTDOWN DISPLAY		
Task dependency 'DISPLAY BUS INFORMATION' - depender: COUNTDOWN DISPLAY - dependees: AVL SYSTEM*		
SoftGoal dependency 'DISPLAY BE READABLE' - depender: PASSENGER - dependees: COUNTDOWN DISPLAY		

Table 6.14: Results of the check 2.3 for the Countdown system (Part 2).

6.2.4 Check 2.4

This check compares the use case actions from the current use case descriptions (see tables 3.1 and 3.2 on pages 25 and 26) with the use case actions coming from the RequisitePro database. The current use case actions are:

C1. The passenger seeks bus information from the Countdown indicator;

C2. The Countdown indicator shows the bus information for the relevant route(s);

C3. The passenger reads bus information from the Countdown indicator;

C4. The passenger recognises which route number(s) will take them the closest to their destination;

C5. The passenger remembers expected arrival time(s) for bus(es) on route(s) of interest;

C6. The passenger uses the bus information to make decisions about their journey;

C7. Every 30 seconds the AVL system updates the Countdown indicator;

C8. The passenger occasionally checks his/her decision when information on the indicator is updated.

The future use case actions are:

F1. The passenger looks at the Countdown display;

F2. The Countdown display shows the bus information for the relevant route(s);

F3. The passenger recognises which route number(s) will take them the closest to their destination;

F4. The passenger remembers expected arrival time(s) for bus(es) on route(s) of interest;

F5. The passenger decides which bus route to use;

F6. Every 30 seconds the AVL system transmits updated bus information to the Countdown display;

F7. Include (Determine bus location and arrival time);

F8. The passenger occasionally checks decision when information on the indicator is updated.

The actions from the first list which do not match one in the second list are actions C1, C3 and C6. Even though C1 and C6 could be considered by the reader as similar to F1 and F5 respectively, the words used in the sentences are not equivalent enough to allow the QROSS-Checker to consider these as matches. In the second list, the actions F1, F5 and F7 do not match any

action in the first list for the same reasons. But the QROSS-Checker also tells us that the action F6 has no correspondence in the first list. This is due to the fact that when it compares the action F6 from the second list with the action C7 in the first one, it finds a percentage of only 53 of identical words, compared to the total number of words in the sentence describing the action 6. To considered them as equivalent, this percentage should be at least 65. However, we did not decrease this threshold not to be likely to match actions which, actually, are not equivalent.

Check 2.4 *All actions in use case descriptions for the current system are candidate actions for use case descriptions of the future system.*		
Do you want to remove these actions to create the new system?		
	Comments from City	Comments from others
1. The passenger seeks bus information from the Countdown indicator		
3. The passenger recognises which route number(s) will take them the closest to their destination		
6. The passenger uses the bus information to make decisions about their journey		
Do you want to add these actions to create the new system?		
	Comments from City	Comments from others
1. The passenger looks at the Countdown display		
5. The passenger decides which bus route to use		
6. Every 30 seconds the AVL system transmits updated bus information to the Countdown display		
7. Include (Determine bus location and arrival time)		

Table 6.15: Results of the check 2.4 for the Countdown system.

6.2.5 Check 2.5

This check is similar to the previous one, except that it compares use case variations instead of actions. The use case variations found in the current use case descriptions (see tables 3.1 and 3.2 on pages 25 and 26) are:

1. If passenger has a mobility restriction, then passenger seeks information about mobility buses from the Countdown indicator (relative to action C1 in section 6.2.4);
2. If wet weather, then passenger may decide not to use a bus if expected waiting time is too long (relative to action C6 in section 6.2.4);
3. If night, then passenger may decide not to use a bus if expected waiting time is too long (relative to action C6 in section 6.2.4).

The use case variations present in the RequisitePro database are:

1. If the passenger has poor eyesight, then he can seek information from the Countdown display in an audible way (relative to action F1 in section 6.2.4);
2. If the AVL system does not update the various Countdown displays with the new expected arrival times, then a passenger message is displayed on the Countdown displays (relative to action F6 in section 6.2.4).

We can easily notice that no correspondence can be found between these two lists. The table provided by the QROSS-Checker thus gives us the expected results.

Check 2.5 *All variations in use case descriptions for the current system are candidate variations or use cases for the future system.*		
Do you want to remove these variations to create the new system?		
	Comments from City	Comments from others
1. If passenger has a mobility restriction, then passenger seeks information about mobility buses from the Countdown indicator		
6. If wet weather, then passenger may decide not to use a bus if expected waiting time is too long		
6. If night, then passenger may decide not to use a bus if expected waiting time is too long		
Do you want to add these variations to create the new system?		
	Comments from City	Comments from others
1. If the passenger has poor eyesight, then he can seek information from the Countdown display in an audible way		
6. If the AVL system does not update the various Countdown displays with the new expected arrival times, then a passenger message is displayed on the Countdown displays		

Table 6.16: Results of the check 2.5 for the Countdown system.

6.2.6 Check 2.6

This check compares the external actors from the future SD model (see figure 3.6 on page 27) with the actors present in the use case description for the future system (see table 3.3 on page 28). In the use case description, the field "Actors" is empty. A line in the table providing the results announce that all the fields corresponding to actors are empty in the use case descriptions.

Check 2.6 *All external actors in the i* SD model of the future system should correspond to actors in the use case descriptions for the future system.*		
All actors fields are empty in the use case descriptions.		
These external actors in the Strategic Dependency model for the current system are not present in the use case descriptions:		
	Comments from City	Comments from others
PASSENGER		
GPS		
LONDON TRANSPORT		
COMMS SYSTEM		

Table 6.17: Results of the check 2.6 for the Countdown system.

6.2.7 Check 2.7

Remember that this check is not implemented according to its statement. Indeed, it provides the list of task dependencies found in the future SD model. These dependencies are represented by hexagons on figure 3.6 on page 27.

Check 2.7 *For all task dependencies identified in the i* SD model of the future system, which represent tasks carried out by actors in the use case diagram, there should be a part of a use case description which describes how those tasks are carried out.*		
Here is a list of the task dependencies in the SD Model for the future system:		
	Comments from City	Comments from others
'CALCULATE ARRIVAL TIMES' - depender: AVL SYSTEM* - dependees: ON-BOARD BUS SYSTEM		
'MAKE TRAVEL DECISIONS' - depender: PASSENGER - dependees: COUNTDOWN DISPLAY		
'DISPLAY BUS INFORMATION' - depender: COUNTDOW DISPLAY - dependees: AVL SYSTEM*		

Table 6.18: Results of the check 2.7 for the Countdown system.

6.2.8 Check 2.8

Once again, for the same reason as for check 1.5, we give the statement of the check in the MS Word document presenting the results of the checks obtained by running the QROSS-Checker.

Check 2.8
All requirements associated with use cases using the RESCUE use case template should be stored in the requirements database.

Table 6.19: Statement of the check 2.8 (not implemented).

Chapter 7

Evaluation and future works

7.1 Evaluation

The work we did brings contributions to several levels.

On the one hand, we can note some contributions on the level of the RESCUE process.
We provide a prototype for tool support needed in the RESCUE process. Indeed, doing the checks "manually" is a great time spending. Moreover, it is possible that human beings make mistakes, and the automation can avoid that.
By redefining the way models and templates had to be built, we have harmonized the use of tools in the RESCUE process. We gave a step-by-step user guide to use these tools, making it easier and avoiding variations in notations, which could otherwise be prone to misunderstanding.
During the development of the QROSS-Checker, we had to make the definitions of the checks clear, to make sure they were implemented the right way. We had to make tacit knowledge such as the meaning of "to be candidate" or "external actors", explicit. We also provided meta models of the various notations. This helped to formalize some parts of the process, and make it more accessible to anyone.

On the other hand, the work we did by extending QROSS was useful to test and debug it. Indeed, we collaborated with the QROSS team at the CETIC, reporting any problem found and consequently improve the tool. Contributions have mainly been brought to the OCL parser and to the metric engine.
Moreover, our work extends the usage and application domain initially intended for QROSS, that is computing metrics on source code.

So, the development of our tool gives evidence of feasibility and usefulness of tool support in the RESCUE process. However, we can enumerate several limitations:

- it is not complete, as some checks have not been implemented;
- it is not very user-friendly. Indeed, we had to constrain users on the way they draw their models, or the sentences they use in use case and requirements templates. Moreover, the use of the tool is a little cumbersome at times, since it is necessary to record the files of the models and the templates under specific formats and to make copies of several tables of RequisitePro's database. It is not really constraining for small projects, but for projects of greater scale, the workload could be consequently weighed down;
- it is not generic: the various parsers and meta models for each type of RESCUE model needed to be implemented by hand in Python;
- the implementation of our QROSS-Checker was tried out only on a small post-mortem case study. It was sufficient for demonstrating and illustrating the purpose of the prototype, but the next improvements of the tool will necessitate more accurate validation, both on projects of larger scale, and undergoing development;
- for checks, where we need to compare two sentences, mainly related to use case and requirements templates (such as the check 1.10), the implementation of the comparisons is approximative and limits the reliability of some results.

7.2 Future Works

Several improvements could thus be considered to overcome these difficulties:

- the first and more obvious improvement would be to implement all the checks for each phase of the RESCUE process;
- to improve the handling of natural language, two kinds of solution can be considered:
  - On the one hand, if we do not want to constraint the users of the RESCUE process, we could chose to use a natural language analyzer. A tool of this type, as the name says, is able to analyze natural language and to extract semantic information out of it. So, it should be able to determine if two sentences used in different requirements templates have the same meaning or not. However, to be really effective, analyzers must possess domain knowledge.
  - On the other hand, if the option to constrain the users can be considered, controlled natural languages could be used.

"Controlled Natural Languages are subsets of natural languages whose grammars and dictionaries have been restricted in order to reduce or eliminate both ambiguity and complexity. Traditionally, controlled languages fall into two major categories: those that improve readability for human readers, particularly non-native speakers, and those that improve computational processing of the text." [Schw]

The use of such a language could help in several ways. First, if the project developed involves people speaking different languages, it could ease the understanding between people. It could also force the users to formalize their requirements and thus it can be a new way to make the requirements even better at an early stage. Obviously, it also makes it easier to compare sentences one with the other.

There are, in the literature, recommendations prescribing to write the contents of the scenarios, use cases and so on in a very simple way, close to a Controlled Natural Language, e. g. [Cock 00].

Bibliography

[Alex 04] I. Alexander and N. Maiden, Eds. *Scenarios, stories, use cases: Through the Systems development Life-Cycle*. John Wiley&Sons, Ltd., 2004.

[Boeh 81] B. W. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.

[Bray 02] I. K. Bray. *An introduction to requirements engineering*. Pearson Education, 2002.

[Cock 00] A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[Heym 01] P. Heymans. *Animating Albert II specifications*. PhD thesis, University of Namur, dec 2001.

[Jone 04] S. Jones and N. Maiden. "RESCUE: An Integrated Method for Specifying Requirements for Complex Socio- Technical Systems". 2004. to appear in: Requirements Engineering for Sociotechnical Systems, J.L. Mate and A. Silva (eds), Idea Group Inc., 2004.

[Maid 03] N. A. M. Maiden, S. Jones, and M. Flynn. "Innovative Requirements Engineering Applied to ATM". In: *ATM'2003 Conference*, 2003.

[Maid 04a] N. A. M. Maiden, S. Manning, S. Jones, and J. Greenwood. "Towards Pattern-Based Generation of Requirements from systems models". In: *International Workshop on Requirements Engineering – Foundations for Software Quality*, 2004.

[Maid 04b] N. Maiden and S. Jones. "An integrated user-centred requirements engineering process". Tech. Rep. VERSION 5, Centre for Human-Computer Interaction, City University, sep 2004. for National Air Traffic services Ltd (NATS).

[Maid 04c] N. Maiden and S. Jones. "RESCUE Process: Examples.". Tech. Rep. 3, Centre for Human-Computer Interaction, City University, sep 2004. for National Ait Traffic services Ltd (NATS).

[Maid 04d] N. A. M. Maiden, S. V. Jones, S. Manning, J. Greenwood, and L. Renou. "Model-Driven Requirements Engineering: Synchronising Models in an Air Traffic Management Case Study.". In: *CAiSE*, pp. 368–383, 2004.

[Maid 05a] N. Maiden. *Requirements Engineering (Module IN3015/INM311)*, Chap. 4, p. 21. City University, 2004-2005.

[Maid 05b] N. Maiden and S. Robertson. "Developing use cases and scenarios in the requirements process". In: *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pp. 561–570, ACM Press, New York, NY, USA, 2005.

[Maid 05c] N. Maiden and S. Robertson. "Integrating Creativity into Requirements Processes: Experiences with an Air Traffic Management System". In: *to appear in: 13th IEEE International Conference on Requirements Engineering (RE 2005)*, IEEE Computer Society, 2005.

[Robe 99] S. Robertson and J. Robertson. *Mastering the Requirements Process*. Addison-Wesley, 1999.

[Rumb 99] J. Rumbaugh, I. Jacobson, and G. Booch, Eds. *The Unified Modeling Language reference manual*. Addison-Wesley Longman Ltd., Essex, UK, UK, 1999.

[Schw] R. Schwitter. "Controlled Natural Languages.". http://www.ics.mq.edu.au/~rolfs/controlled-natural-languages/. Centre for Language Technology, Macquarie University's.

[Sutc 02] A. Sutcliffe. *User-Centred requirements engineering: Theory and practice*. Springer, 2002.

[Sutc 98] A. G. Sutcliffe, N. A. M. Maiden, S. Minocha, and D. Manuel. "Supporting Scenario-Based Requirements Engineering.". *IEEE Trans. Software Eng.*, Vol. 24, No. 12, pp. 1072–1088, 1998.

[Vice 99] K. Vicente. *Cognitive work analysis*. Lawrence Erlbaum Associates, 1999.

[Yu 94] E. S. K. Yu and J. Mylopoulos. "Understanding "Why" in Software Process Modelling, Analysis, and Design.". In: *ICSE*, pp. 159–168, 1994.

[Yu 96] E. S.-K. Yu. *Modelling strategic relationships for process reengineering.* PhD thesis, University of Toronto, Toronto, Ont., Canada, Canada, 1996.

[Yu 97] E. S. K. Yu. "Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering.". In: *RE*, pp. 226–235, 1997.

Appendix A

Statements of the checks

Check 1.1 Every actor in the context model for the current system is a candidate actor for the context model of the future system.

Check 1.2 Every data flow in the context model for the current system is a candidate data flow for the context model of the future system.

Check 1.3 Every use case in the use case diagram for the current system is a candidate for a use case in the use case diagram of the future system.

Check 1.4 Every adjacent actor which communicates directly with the technical system (level 1) in the context model for the current system is a candidate actor for the use case diagram of the current system.

Check 1.5 The system boundary in the use case model for the current system should be the same as the boundary between level 1 and 2 in the context model for the current system.

Check 1.6 For every data flow from or to level 1 (or one of the sub systems within level 1) of the context model of the current system, there should be a corresponding line in the use case diagram for the current system indicating involvement of the relevant actor in the relevant use case.

Check 1.7 Every adjacent actor (at level 2, 3 or 4) which communicates directly with the technical system (level 1) in the context model for the future system is a candidate actor for the use case diagram for the future system.

Check 1.8 The system boundary in the use case model for the future system should be the same as the boundary between level 1 and 2 in the context model for the future system.

Check 1.9 For every data flow from or to level 1 (or one of the sub systems within level 1) of the context model for the future system, there should

be a corresponding line in the use case diagram for the future system indicating involvement of the relevant actor in the relevant use case.

Check 1.10 Services and functions related to use cases in the use case diagram for the future system should map to system level requirements, i.e. high level functional and non functional requirements in the requirements database.

Check 2.1 All external actors in the i* SD model of the current system should correspond to actors in the use case descriptions for the current system.

Check 2.2 All external actors in the i* SD model of the current system are candidate actors for the i* SD model for the future system.

Check 2.3 All dependencies in the i* SD model of the current system are candidate dependencies for the i* SD model for the future system.

Check 2.4 All actions in the use case descriptions for the current system are candidate actions for the use case descriptions for the future system.

Check 2.5 All variations in the use case descriptions for the current system are candidate variations or use cases for the future system.

Check 2.6 All external actors in the i* SD model of the future system should correspond to actors in the use case descriptions for the future system.

Check 2.7 For all task dependencies identified in the i* SD model of the future system, which represent tasks carried out by actors in the use case diagram, there should be a part of a use case description which describes how those tasks are carried out.

Check 2.8 All requirements associated with use cases using the RESCUE use case template should be stored in the requirements database.

Check 4.1 Ensure that each requirement is either a system level requirement or is linked to a use case, use case action, or alternative course.

Check 4.2 Check whether each use case action is linked with requirements of the right types using simple heuristics based on action- and requirement- types, for example do human-computer interaction actions have candidate functional, usability, look-and-feel, training and performance requirements specified for them?

Check 4.3 For all except system-level requirements, check that requirement fit criteria are grounded in the use cases to which requirements are linked.

Check 5.1 Ensure that all impact consequences are recorded in the requirements database.

Check 5.2 Ensure that change requests are generated for all impact consequences recorded in the requirements database.

Appendix B

How to correctly use the QROSS-Checker

B.1 Creating a Context Model

To create this model, you have to use Microsoft Visio. Here is how to draw each of the elements of the context model:

- Domain borders: use the rectangle shape, from the toolbar
- Agents: use the "state" shape of the UML Statecharts you will find on the left of the visio window, and give a name to the shape created
- System (or technical) agents: use the ellipse shape, from the toolbar, and give a name to the shape created
- Information: use the "message" arrow of the UML Sequence, and give it a name. Be careful that a red rectangle appears around the actors you want to connect to each other, when you bring one of the ends of the arrow in it. If several informations are transmitted between the same actors, separate the name of each information by a "/" when you give a name to the arrow (cfr "Instructions/status" on figure B.1).

B.2 Creating a Use Case Diagram

Once again, you have to use Microsoft Visio to create this diagram. Here is the method to do it:

- Use case Actor: use the "actor" shape of the UML Use case tool, and give it a name.
- Connections between an actor and a use case: use the "communication" arrow from the UML Use Case tool.

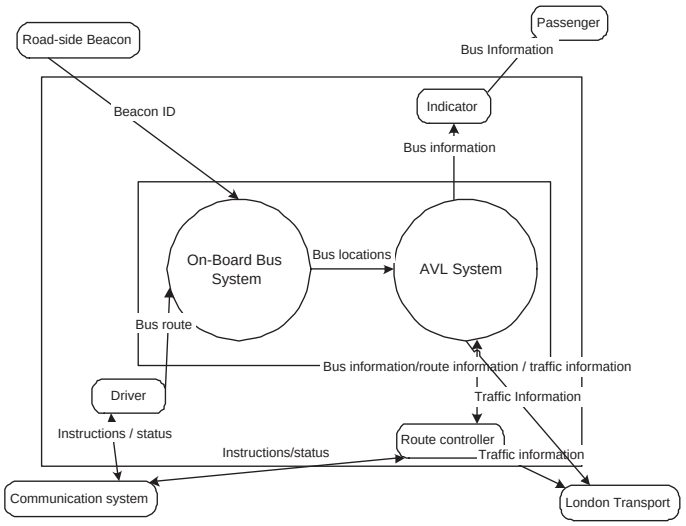

Figure B.1: Context Model example

- Borders: use the System boundary from the UML Use Case tool.
- Extends relationships: use the 'extends' arrow from the UML Use Case tool.

When connecting an actor and a use case with a communication arrow, or two use cases with an extend arrow, be careful that a red rectangle appears around the elements you're connecting.

B.3 Creating a Strategic Dependency Model

Here, you have to use the Redepend tool. When you create a dependency, you have to be careful: on each actor shape, you will find several connection points; to each connection point, only one dependency can be connected. If you need more connection points than the number available for an actor, you have to "duplicate" this actor, i.e. create a new actor, with exactly the same name. Let us be careful as well to the fact that the connection point becomes red each time you connect a dependency to it. To differentiate the "internal" (i.e. non external) actors in the schema, put a "*" in the name of these. Do not use the character "*" in others actor's names.

B.4 Using the QROSS-Checker to do the synchronization checks

First, in "~\qross\data\import" (where ~ is the root file where you have installed the qross files), you will find a "public" file. In this file, to make

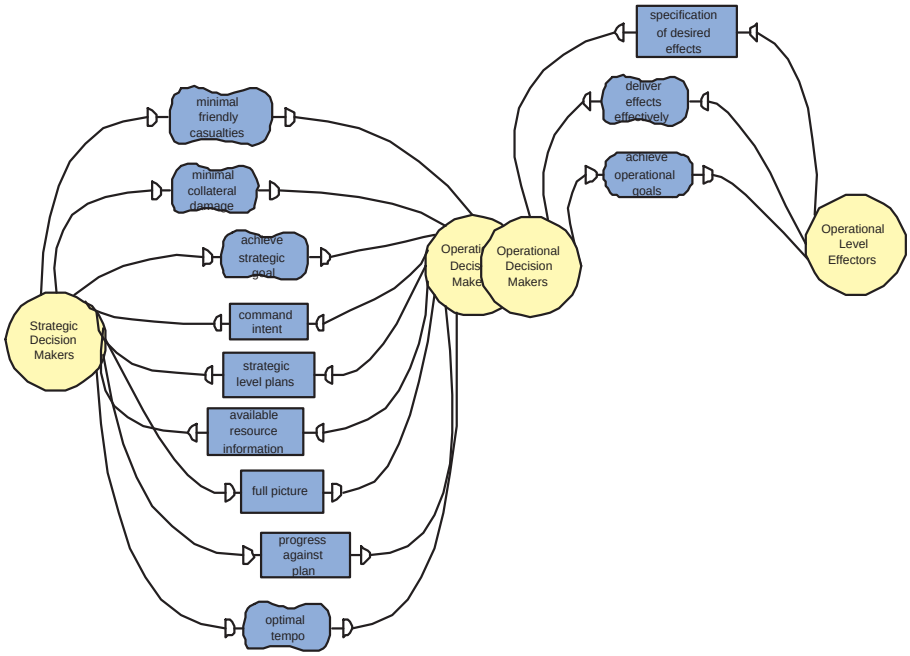

Figure B.2: Strategic dependency model example

the task easier, create another file which you name with the name of your project, let us say “MyProject”. Then, save your context model in visio as an XML Drawing file (“.vdx”). The name you give to it must contain, at least “ContextModel”. If it is the context model for the current system, the name has to contain the word ’Current’, and if it’s for the future system, the name has to contain the word ’Future’. For instance, if your project is named BusDriver, the file containing the context model for the current system can be named “BusDriverCurrentContextModel.vdx”. Save it in the ’public’ subfile of qross, “MyProject”. Follow the same process for your Use Case Diagram and your Strategic Dependency Model, but the parts which have to be present in the names are “UseCase” and “SDModel”, respectively. Then, save the word documents containing the use case descritpions as web pages (“.htm”) in the public file “MyProject”. And finally, here is the procedure to give to qross the access to the requirements of your project. Go to your “Control Panel”, then double click on “administration tools”, then on “data sources (ODBC)”, add a “Microsoft Access Driver (*.mdb)”, give it the name you want and select the database containing the requirements of your project. Open the database in Microsoft Access, and copy the tables “RqRequirements”, “RqRequirementTypes”, “RqUserDefinedFields”, “RqUserDefinedFieldValues”, “RqPackages” and “RqPackageElement” by following this method:

- Right click on the name of the table, and choose “save as”;

- The name of the copy of the table will be the same as the former one, just add a "2" at the end ("RqRequirements" will become "RqRequirements2");
- Click on "ok".

Finally, in the public file "MyProject", create a text file named "Requirements.txt". Write the name you have given to the ODBC data source, nothing else, and with exactly the same orthography.

B.4.1 Launching the QROSS-Checker

- Each time you use the QROSS-Checker, you have to delete the four "zodb" files in "~\qross\data\db" before starting the application;
- Go in "~\qross\run" and dubble click on each of the '.bat' files;
- Start your internet browser and enter the adress "http://localhost:9081/
- Click on "project";
- Log in with "admin" as a login and "admin" as a password;
- Click on "new", enter a name for your project on the right place, click on the file icon near "public";
- Select the file of your project and click on "create";
- You'll find the result of the synchronization checks in "~\qross\data\projects\Myproject\result\SynchronizationChecks.doc".

Appendix C

Requirements types

Requirement type	Types of units of measure
FR - Functional requirement	Requirements state predicted outcomes from functions, therefore test for accurate completion of function within standard.
AS - Applicable standards	Identified standard that is applied to the future system and its development.
AR - Availability requirements	Periods of time to be available, percentage of permitted downtime in a period, maximum allowed duration of a downtime, numbers of separate unavailable times. May be supplemented by definition of what is unavailability of a system or product.
DR - Device requirements	Adherence to some external standard, normally tested by 3rd party expert combined with observation of the product.
IOR - Interoperability requirements	Other systems to operate with for particular functions.
LFR - Look and Feel requirements	Adherence to some external standard, normally tested by 3rd party expert.
MR - Maintenance requirements	Time needed to maintain, resources needed to maintain, levels or types of maintenance to support.
OR Organisational requirements	Organisational structures such as reporting hierarchies or lines of communication.
PR - Performance requirements	Speed measured using response-times or times to complete an action, and throughput measured as actions or functions undertaken within a time period.
PCR Political/cultural requirements	Customs or cultural norms within which the system must operate.
PTR - Portability requirements	Platforms that the system needs to operate on.
RCR - Recoverability requirements	Mean-time to recover, likelihood to recover.
RR - Reliability requirements	Mean-time between failures.
SFR - Safety requirements	Number of injuries/risk of injuries total or over time.
SR - Security requirements	Access functions, mean-time between breaches.
TR - Training requirements	Extent, length and nature of user training needed, and measure of success of the training.
UR - Usability requirements	Task-completion times, error-rates, usage rates.

Table C.1: Measures for different requirement types from [Maid 04a]

Appendix D

Code for parsers

D.1 Parsers: general methods

```
# General Parser Class ————————————————————-
# Contains the methods shared by all the parsers ——————————
# ————————————————————————————————
class Parser:
    def _init_(self):
        pass

    def ParseMaster(self, line, file):
        masters =
        while line.find('</Masters>') == -1:
            if line.find('</Masters>') == -1:
                if line.find('<Master ') == -1:
                    line = self._Find('<Master ', file)
                id = self._GetValueOf('ID=\", line, '\' NameU=')
                name = self._GetValueOf('NameU=\", line, '\")
                masters[id] = name
                line = self._Find('</Master>', file)
                line = file.readline()
            else: break
        return masters

    def _Find(self, arg, file):
        l = file.readline()
        while l.find(arg)==-1:
            l = file.readline()
        return l

    def _GetValueOf(self, arg, string, endCharacter):
```

```
        # arg = part of the string which precedes the value to get
        # string = string in which searching
        # endCharacter = character following the value to get
        # return the value
        i = string.find(arg, 0, len(string))+len(arg)
        if i == -1+len(arg):
            return 'error in _GetValueOf'
        else:
            j = string.find(endCharacter, i, len(string))
            if j ==-1:
                return 'error in _GetValueOf'
            else:
                return string [i:j]

    def _GetValueOf2(self, arg, string, endCharacter, f):
        # arg = part of the string which precedes the value to get
        # string = string in which searching
        # endCharacter = character following the value to get
        # return the value
        i = string.find(arg, 0, len(string))+len(arg)
        if i == -1+len(arg):
            return 'error in _GetValueOf'
        else:
            j = string.find(endCharacter, i, len(string))
            while j ==-1:
                string = string + f.readline()
                j = string.find(endCharacter, i, len(string))
            else:
                return string [i:j]

    def _SearchInLine(self, line, file):
        i = 0 # counter for line
        j = 0 # counter for res
        k = 0 # number of '<' characters, unclosed
        res = ''
        num = None
        bool = 0
        while i != len(line)+1:
            if i == len(line):
                line = line + file.readline()
                d = line.find('<![if !supportLists]>')
                if d!=-1:
                    c = 1
                    while c==1:
```

```
                    line = line + file.readline()
                    f = line.find('<![endif]>')
                    if f!=-1:
                        c = 0
                        num = self._SearchInLine(line, file)
                        line = line[:d]+line[f+10:]
    if line[i] == '<':
        if res == '':
            bool = 1
            k = k+1
        else:
            break
    elif line[i] == '>':
        k = k-1
    elif (line[i] == '\n') & (res != '') & (k == 0):
        res = res + ' '
        j=j+1
    else:
        if (k == 0) & (line[i] != '<') & (bool == 1):
            if (res !=''):
                if (res[j-1] == ' ') & (line[i] == ' '):
                    pass
                else:
                    res = res + line[i]
                    j = j+1
            else:
                res = res + line[i]
                j = j+1
    i = i+1
if num!=None:
    return num+' '+res
else:
    return res
```

D.2 Parser for context models

```
#-----------------------------------------------------------------------
# Class parsing an XML file, which contains the description of a context
# model schema
# Imports
from qross.shared.ContextModelElement import *
from qross.shared.Constants import GC
from qross.project_server.Parser import *
import os

class ContextParser(Parser):
    def _init_(self, filepath):
        Parser._init_(self)
        self.schemaName = filepath.split(os.sep)[-1].split('.')[0]
        self.positions =
        # dictionary: Key = Agent, values = [PinX, PinY, Heigth, weight]
        self.domains =
        # dictionary: Key = domain, values = [PinX, PinY, Heigth, weight]
        self.connections =
        # dictionnary: Key = Connection between 2 Agents,
        # values = single right, single left or double
        self.agents =
        # dictionary: Key = id of an Agent, values = name of the Agent
        self.agentsTypes =
        self.elements =
        # dictionary: key: id, values: ContextModelElement
        self.objects =
        # dictionary: Key = id of the object, Values = object in the repository
        self.objectsTypes =
        # dictionary: Key = id of the object, Values = type of UserNeedElement
        f = file(filepath,'r')
        self.parse(f)
        f.close()

    def parse(self, f):
        #l = Parser._Find(self, '<Title>', f)
        # We have find the name of the schema
        # -¿ creation of a schema :)
        #self.schemaName = Parser._GetValueOf(self, '<Title>', l, '<')
        if self.schemaName.upper().find('CURRENT') != -1:
            self.schemaName = 'CurrentContextModel'
        elif self.schemaName.upper().find('FUTURE') != -1:
            self.schemaName = 'FutureContextModel'
```

```
else:
    print 'ERROR in ContextModel file Name'
    print 'maybe you forgot to name it with ContextModel in it'
    print 'or maybe you forgot to put Current or Future in the file\'s name'
schema = ContextSchema(self.schemaName)
self.objects['schema'] = schema
self.objectsTypes['schema'] = GC.ART_UNE_SCHEMA
l= Parser._Find(self, '<Masters>', f)
self.masters = Parser.ParseMaster(self, l, f)
l = Parser._Find(self, '<Shapes>', f)
l = f.readline()
# Threatement of the shapes
i=0
while l.find('</Shapes>') == -1:
    if l.find('<Shape')==-1:
        l = f.readline()
    if l.find('</Shapes>') != -1:
        break
    if l.find('<Shape')!=-1:
        id = Parser._GetValueOf(self,'ID=', l, ' ')
        if self.schemaName == 'FutureContextModel':
            id = 'F'+ id
        if l.find('Master=\'18\"') != -1:
            type = '18'
        else:
            type = Parser._GetValueOf(self, 'Master=\", l, '\")
        if type == 'error in _GetValueOf':
            l = self.Shape(l, id, f, self.domains, schema)
        elif self.masters[type].find('Message')!=-1:
            l = self.Information(l, id, f, self.connections, schema)
        elif self.masters[type].find('State')!=-1:
            l = self.Agent(l, id, f, self.positions, self.objects, schema)
        else:
            pass
        j = 1
        if ((type != 'error in _GetValueOf')and
            (self.masters[type].find('State')!=-1)):
                l = f.readline()
        if l.find('</Shape>') != -1:
            j=j-1
        if l.find('<Shape ') !=-1:
            l = f.readline()
        while j != 0:
            l = f.readline()
```

```
                    if l.find('<Shape ')!=-1:
                        j = j+1
                    elif l.find('</Shape>') != -1:
                        j = j-1
                i = i+1
        self.ComputeDomains()
        self.ComputePositions()
        l = Parser._Find(self, '<Connects>', f)
        l = f.readline()
        while l.find('</Connects>') == -1:
            self.Connection(l, self.connections, f)
            l = f.readline()

    def Shape(self, line, id, file, domains, schema):
        l= Parser._Find(self, '<PinX>', file)
        x = float(self._GetValueOf('<PinX>', l, '¡'))
        l =Parser._Find(self, '<PinY>', file)
        y = float(self._GetValueOf('<PinY>', l, '¡'))
        l = Parser._Find(self, '<Width>', file)
        width = float(self._GetValueOf('<Width>', l, '¡'))
        l = Parser._Find(self, '<Height>', file)
        height =float( self._GetValueOf('<Height>', l, '<'))
        name = ''
        while l.find('</Shape>') == -1:
            if l.find('<Text>') == -1:
                l=file.readline()
            else:
                # creation of a TechnicalAgent which the name is get in this line
                if l.find('><')==-1:
                    name = Parser._SearchInLine(self, l, file)
                else:
                    name = Parser._SearchInLine(self, l, file)
                object = ContextTechnicalAgent(id, name.upper())
                object.SetSchema(self.objects['schema'])
                self.elements[id] = object
                self.objects[id] = object
                self.objectsTypes[id] = GC.ART_UNE_SCHEMAPIECE
                break
        if name == '':
            if domains.has_key('first'):
                if domains.has_key('second'):
                    domains['third'] = (x,y,width, height)
                else:
                    domains['second'] = (x, y, width, height)
```

```
            else:
                domains['first'] = (x, y, width, height)
        else:
            self.agentsTypes[id] = 'TechnicalAgent'
        return l

    def Information(self, line, id, file, connections, schema):
        l = Parser._Find(self, '<Line>', file)
        i = '0'
        j='0'
        while l.find('</Line>')==-1:
            while l.find('<BeginArrow')==-1:
                l = file.readline()
            i = Parser._GetValueOf(self, '>', l, '</')
            l = file.readline()
            j = Parser._GetValueOf(self, '>', l, '</')
            break
        if (i == '0') & (j != '0'):
            connections[id] = 'singleright'
        elif (i != '0') & (j == '0'):
            connections[id] = 'singleleft'
        elif (i != '0') & (j != '0'):
            connections[id] = 'double'
        elif (i == '0') & (j == '0'):
            print 'problem in Information'
        l = Parser._Find(self, '<Text>', file)
        information = Parser._SearchInLine(self, l, file)
        # creation of an object 'ContextInformation'
        object = ContextInformation(information.upper(), id, schema)
        object.SetSchema(self.objects['schema'])
        self.elements[id] = object
        self.objects[id] = object
        self.objectsTypes[id] = GC.ART_UNE_SCHEMAPIECE
        return l

    def Agent(self, line, id, file, positions, objects, schema):
        l = Parser._Find(self, '<PinX>', file)
        x = float(self._GetValueOf('<PinX>', l, '¡'))
        l =Parser._Find(self, '<PinY>', file)
        y = float(self._GetValueOf('<PinY>', l, '<'))
        l = Parser._Find(self, '<Width ', file)
        width = float((self._GetValueOf('<Width ', l, '<').split('>')[1]).split('<')[0])
        l = Parser._Find(self, '<Height Unit=\'MM\' F=\'Inh\'>', file)
        height = float(self._GetValueOf('<Height Unit=\'MM\' F=\'Inh\'>', l, '<'))
```

```
        l = Parser._Find(self, '<Shapes>', file)
        i = 0
        while i != 3:
            l = Parser._Find(self, '<Shape', file)
            i = i+1
        l = Parser._Find(self, '<Text>', file)
        nameAgent = Parser._SearchInLine(self, l, file)
        self.agents[id] = nameAgent.upper()
        self.positions[id]=[x, y, width, height]
        l = Parser._Find(self, '</Shapes>', file)
        return l

    def Connection(self, line, connections, file):
        id = Parser._GetValueOf(self, 'FromSheet=', line, ' F')
        sender = Parser._GetValueOf(self, 'ToSheet=', line, ' T').upper()
        line = file.readline()
        receiver = Parser._GetValueOf(self, 'ToSheet=', line, ' T').upper()
        if self.schemaName == 'FutureContextModel':
            id = 'F'+ id
            sender = 'F' + sender
            receiver = 'F' + receiver
        direction = connections[id]
        if direction == 'singleright':
            # create a 'send' relationship between the ContextAgent sender and
            # the ContextInformation id
            sender = self.objects[sender]
            information = self.objects[id]
            sender.AddSentInformation(information)
            information.AddSender(sender)
            #create a 'receive' relationship between the ContextInformation id
            # and the ContextAgent receiver
            receiver = self.objects[receiver]
            information.AddReceiver(receiver)
        elif direction == 'singleleft':
            # create a 'send' relationship between the ContextAgent receiver and
            # the ContextInformation id
            sender = self.objects[receiver]
            information = self.objects[id]
            sender.AddSentInformation(information)
            information.AddSender(sender)
            # create a 'receive' relationship between the ContextInformation id
            # and the ContextAgent sender
            receiver = self.objects[sender]
            information.AddReceiver(receiver)
```

```
        else:
            # bidirectional relationship
            sender = self.objects[sender]
            information = self.objects[id]
            receiver = self.objects[receiver]
            sender.AddSentInformation(information)
            information.AddSender(sender)
            information.AddReceiver(receiver)
            receiver.AddSentInformation(information)
            information.AddSender(receiver)
            information.AddReceiver(sender)

    def ComputeDomains(self):
        if self.domains.has_key('first') & self.domains.has_key('second'):
            a=self.domains['first']
            b=self.domains['second']
            c=[]
            if self.domains.has_key('third'):
                c = self.domains['third']
            if (a[2] < b[2]) & (a[3] < b[3]):
                if c != []:
                    if (c[2] < b[2]):
                        if (c[2] < a[2]):
                            self.domains['third']=b
                            self.domains['second']=a
                            self.domains['first']=c
                        else:
                            self.domains['third']=c
                            self.domains['second']=b
                    else:
                        # domains were well ordered
                        pass
                if c == []:
                    # domains were well ordered
                    pass
            else:
                if c != []:
                    if (c[2] < a[2]):
                        if c[2] < b[2]:
                            self.domains['third']=a
                            self.domains['second']=b
                            self.domains['first']=c
                        else:
                            self.domains['third']=a
```

```
                                self.domains['second']=c
                                self.domains['first']=b
                        else:
                            self.domains['third']=b
                            self.domains['second']=a
                            self.domains['first']=c
                    else:
                        self.domains['second']=a
                        self.domains['first']=b
            else:
                print 'error in domains @ComputeDomains'

    def ComputePositions(self):
        for key in self.agents.keys():
            if self.Include(self.positions[key], self.domains['first']):
                self.agentsTypes[key] = 'TechnicalAgent'
                agent = ContextTechnicalAgent(key, self.agents[key])
                agent.SetSchema(self.objects['schema'])
                self.objects[key] = agent
                self.objectsTypes[key] = GC.ART_UNE_SCHEMAPIECE
            elif self.Include(self.positions[key], self.domains['second']):
                self.agentsTypes[key] = 'SocioTechnicalAgent'
                agent = ContextSocioTechnicalAgent(key, self.agents[key])
                agent.SetSchema(self.objects['schema'])
                self.objects[key] = agent
                self.objectsTypes[key] = GC.ART_UNE_SCHEMAPIECE
            elif self.domains.has_key('third'):
                if self.Include(self.positions[key], self.domains['third']):
                    self.agentsTypes[key] = 'UncontrolledAgent'
                    agent = ContextUncontrolledSystemAgent(key, self.agents[key])
                    agent.SetSchema(self.objects['schema'])
                    self.objects[key] = agent
                    self.objectsTypes[key] = GC.ART_UNE_SCHEMAPIECE
                else:
                    self.agentsTypes[key] = 'OutsideAgent'
                    agent = ContextOutsideAgent(key, self.agents[key])
                    agent.SetSchema(self.objects['schema'])
                    self.objects[key] = agent
                    self.objectsTypes[key] = GC.ART_UNE_SCHEMAPIECE
            else:
                self.agentsTypes[key] = 'OutsideAgent'
                agent = ContextOutsideAgent(key, self.agents[key])
                agent.SetSchema(self.objects['schema'])
                self.objects[key] = agent
```

```
                self.objectsTypes[key] = GC.ART_UNE_SCHEMAPIECE

    def Include(self, object, domain):
        domainX = domain[0]
        domainY = domain[1]
        domainWidth = domain[2]
        domainHeight = domain[3]
        objectX = object[0]
        objectY = object[1]
        if ((domainX - domainWidth/2) < objectX) &
            (objectX < (domainX + domainWidth/2)) &
            ((domainY - domainHeight/2) < objectY) &
            (objectY < (domainY + domainHeight/2)):
                return 1
        else:
            return 0
```

Appendix E

Code for the checks

```
#@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
# Requests to implement the Synchronisation checks
# Check 1.1 **************************************************
# Every actor in the context model for the current system is a candidate
# actor for the context model of the future system ***********
CurrentContextAgents = currentContextModel.pieces
   →select(agent:ContextAgent|agent.type = 'agent').GetName2()
FutureContextAgents = futureContextModel.pieces
   →select(agent:ContextAgent | agent.type = 'agent').GetName2()
Check_1_1_Part1 = CurrentContextAgents
   →select(actor|not FutureContextAgents→includes(actor))
Check_1_1_Part2 = FutureContextAgent
   →select(actor|not CurrentContextAgents→includes(actor))
# Check 1.2 **************************************************
# Every Data flow in the context model for the current system is a candidate
# data flow for the context model of the future system ***********
CurrentDataFlow = currentContextModel.pieces
   →select(info:ContextInformation|info.type = 'info').GetCompleteInfo()
FutureDataFlow = futureContextModel.pieces
   →select(info:ContextInformation|info.type = 'info').GetCompleteInfo()
Check_1_2_Part1 = CurrentDataFlow
   →select(info|not FutureDataFlow→includes(info))
Check_1_2_Part2 = FutureDataFlow
   →select(info|not CurrentDataFlow→includes(info))
# Check 1.3 **************************************************
# Every use case in the use case diagram for the current system is a candidate
# for a use case in the use case diagram for the future system ****
CurrentUseCases = currentUseCaseDiagram.pieces
   →select(uc:UseCase | uc.__class__.__name__ = 'UseCase').GetName2()
FutureUseCases = futureUseCaseDiagram.pieces
```

```
  →select(uc:UseCase | uc.__class__.__name__ = 'UseCase').GetName2()
Check_1_3_Part1 = CurrentUseCases
  →select(uc|not FutureUseCases→includes(uc))
Check_1_3_Part2 = FutureUseCases
  →select(uc|not CurrentUseCases→includes(uc))
# Check 1.4 ************************************************
# Every adjacent actor which communicates directly with the technical system
# (level 1) in the context model for the current system is a candidate
# actor for the use case diagram for the current system *********
CurrentLevel1 = currentContextModel.pieces
  →select(actor | actor.__class__.__name__ = 'ContextTechnicalAgent')
CurrentLevel234 = currentContextModel.pieces
  →select(actor | actor.type = 'agent' and
  not CurrentLevel1→includes(actor)).GetName2()
CurrentUseCaseActors = currentUseCaseDiagram.pieces
  →select(actor | actor.__class__.__name__ = 'UseCaseActor').GetName2()
CurrentDataFlows = currentContextModel.pieces
  →select(df | df.__class__.__name__ = 'ContextInformation')
CurrentAdjDataFlows = CurrentDataFlows→select(df|
  ((CurrentLevel1→includes(df.GetFirstSender()) and
  (not CurrentLevel1→includes(df.GetFirstReceiver()))) or
  (CurrentLevel1→includes(df.GetFirstReceiver()) and
  (not CurrentLevel1→includes(df.GetFirstSender()))))
CurrentAdjActors = CurrentAdjDataFlows.GetAllSenders().GetName2()
  →union(CurrentAdjDataFlows.GetAllReceivers().GetName2())
  →select(actor|CurrentLevel234→includes(actor))→asSet()
Check_1_4_Part1 = CurrentAdjActors
  →select(actor|not CurrentUseCaseActors→includes(actor))
Check_1_4_Part2 = CurrentUseCaseActors
  →select(actor|not CurrentAdjActors→includes(actor))
# Check 1.5 ************************************************
# The system boundary in the use case model for the current system model
# should be the same as the boundary between levels 1 & 2 in the context
# model for the current system *********************************
  # Non implemented
# Check 1.6 ************************************************
# For every data flow from or to level 1 ( or one of the sub systems within
# level 1) of the context model of the current system there should be a
# corresponding line in the use case diagram for the current system indicating
# involvement of the relevant actor in the relevant use case ****
CurrentUseCaseInfo = currentUseCaseDiagram.pieces
  →select(actor:UseCaseActor | actor.type = 'actor').GetCompleteInfo()
CurrentAdjDataFlowsInfo = CurrentAdjDataFlows.GetCompleteInfo()
# Check 1.7 ************************************************
```

```
# Every adjacent actor (at level 2 3 or 4) which communicates directly
# with the technical system (level 1) in the context model for the future
# system is a candidate actor for the use case diagram for the future system*
FutureLevel1 = futureContextModel.pieces
  →select(actor | actor.__class__.__name__ = 'ContextTechnicalAgent')
FutureLevel234 = futureContextModel.pieces
  →select(actor | actor.type = 'agent' and
  not FutureLevel1→includes(actor)).GetName2()
FutureUseCaseActors = futureUseCaseDiagram.pieces
  →select(actor:UseCaseActor | actor.type = 'actor').GetName2()
FutureDataFlows = futureContextModel.pieces
  →select(df | df.__class__.__name__ = 'ContextInformation')
FutureAdjDataFlows = FutureDataFlows→select(df|
  ((FutureLevel1→includes(df.GetFirstSender()) and
  (not FutureLevel1→includes(df.GetFirstReceiver()))) or
  (FutureLevel1→includes(df.GetFirstReceiver()) and
  (not FutureLevel1→includes(df.GetFirstSender()))))
FutureAdjActors = FutureAdjDataFlows.GetAllSenders().GetName2()
  →union(FutureAdjDataFlows.GetAllReceivers().GetName2())
  →select(actor|FutureLevel234→includes(actor))→asSet()
Check_1_7_Part1 = FutureAdjActors
  →select(actor|not FutureUseCaseActors→includes(actor))
Check_1_7_Part2 = FutureUseCaseActors
  →select(actor|not FutureAdjActors→includes(actor))
# Check 1.8 ***************************************************
# The system boundary in the use case model for the future system should
# be the same as the boundary between levels 1 & 2 in the context model
# for the future system ***************************************
  # Non implemented
# Check 1.9 ***************************************************
# For every data flow from or to level 1 ( or one of the sub systems
# within level 1) of the context model of the future system there
# should be a corresponding line in the use case diagram for the future
# system indicating involvement of the relevant actor in the relevant
# use case ****************************************************
FutureUseCaseInfo = futureUseCaseDiagram.pieces
  →select(actor:UseCaseActor | actor.type = 'actor').GetCompleteInfo()
FutureAdjDataFlowsInfo = FutureAdjDataFlows.GetCompleteInfo()
# Check 1.10 **************************************************
# services and functions related to use cases in the use case diagram for
# the future system should map to system level requirements i.e. high level
# functional and non functional requirements in the requirements database *
  # not implemented
# Check 2.1***************************************************
```

```
# All external actors in the i* SD model of the current system should
# correspond to actors in the use case descriptions for the current
# system ******************************************************
CurrentExternalSDActors = currentSDModel.pieces
  →select(actor | actor.__class__.__name__ = 'SDActor')
  →select(actor | actor.IsExternal()¡¿0).GetName2()
CurrentUseCaseDescrActors = currentUseCaseDiagram.pieces
  →select(useCase:UseCase | useCase.type = 'useCase').GetActors()→asSet()
Check_2_1 = CurrentExternalSDActors
  →select(act|not CurrentUseCaseDescrActors→includes(act))
# Check 2.2 ***************************************************
# All external actors in the i* SD model of the current system are candidate
# actors for the i* SD model of the future system *********************
FutureSDActors = futureSDModel.pieces
  →select(actor | actor.__class__.__name__ = 'SDActor').GetName2()
Check_2_2_Part1 = CurrentExternalSDActors
  →select(act|not FutureSDActors→includes(act))
Check_2_2_Part2 = FutureSDActors
  →select(act|not CurrentExternalSDActors→includes(act))
# Check 2.3 ***************************************************
# All dependencies in the i* SD model of the current system are candidate
# dependencies for the i* SD model of the future system ***************
CurrentSDDependency = currentSDModel.pieces
  →select(dep:SDDependency | dep.__class__.__name__ = 'SDResourceDependency'
  or dep.__class__.__name__ = 'SDTaskDependency' or dep.__class__.__name__ =
  'SDGoalDependency' or dep.__class__.__name__ = 'SDSoftGoalDependency')
  .GetCompleteInfo()
FutureSDDependency = futureSDModel.pieces
  →select(dep:SDDependency | dep.__class__.__name__ = 'SDResourceDependency'
  or dep.__class__.__name__ = 'SDTaskDependency' or dep.__class__.__name__ =
  'SDGoalDependency' or dep.__class__.__name__ = 'SDSoftGoalDependency')
  .GetCompleteInfo()
Check_2_3_Part1 = CurrentSDDependency
  →select(dep|not FutureSDDependency→includes(dep))
Check_2_3_Part2 = FutureSDDependency
  →select(dep|not CurrentSDDependency→includes(dep))
# Check 2.4 ***************************************************
# All actions in use case descriptions for the current system are
# candidate actions for use case descriptions of the future system *******
CurrentUseCaseActions = currentUseCaseDiagram.pieces
  →select(uc:UseCase | uc.__class__.__name__ = 'UseCase').GetActions()
FutureUseCaseActions = futureUseCaseTable.pieces
  →select(uc:UseCase | uc.__class__.__name__ = 'UseCase').GetActions()
# Check 2.5 ***************************************************
```

```
# All variations in use case descriptions for the current system are
# candidate variations or use cases for the future system ************
CurrentUseCaseVariations = currentUseCaseDiagram.pieces
  →select(uc:UseCase | uc.__class__.__name__ = 'UseCase').GetVariations()
FutureUseCaseVariations = futureUseCaseTable.pieces
  →select(uc:UseCase | uc.__class__.__name__ = 'UseCase').GetVariations()
# Check 2.6 **************************************************
# All external actors in the i* SD model of the future system should
# correspond to actors in the use case descriptions for the future system *
FutureExternalSDActors = futureSDModel.pieces
  →select(actor | actor.__class__.__name__ = 'SDActor')
  →select(actor | actor.IsExternal()<>0).GetName2()
FutureUseCaseDescrActors = futureUseCaseDiagram.pieces
  →select(useCase:UseCase | useCase.type = 'useCase').GetActors()→asSet()
Check_2_6 = FutureExternalSDActors
  →select(act|not FutureUseCaseDescrActors→includes(act))
# Check 2.7 **************************************************
# For all task dependencies identified in the i* SD model of the future
# system which represent tasks carried out by actors in the use case diagram
# there should be a part of a use case description which describes how
# those tasks are carried out*************************************
FutTaskDependencies = futureSDModel.pieces
  →select(dep:SDDependency | dep.__class__.__name__ = 'SDTaskDependency')
  .GetCompleteInfo()
# Check 2.8 **************************************************
# All requirements associated with use cases using the RESCUE use case
# template should be stored in the requirements database *********
ReqUCDB = futureUseCaseTable.pieces
  →select(req | req.__class__.__name__ = 'Requirement').GetReq()
ReqDB = futureReqTable.pieces
  →select(req | req.__class__.__name__ = 'Requirement').GetUserNeedName()
# End requests to implement the Synchronisation checks
# @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


# classgenerating the MS Word file containing the tables
# presenting the results class Metriques:
  def __init__(self, projectObjects):
    self.projectObjects = projectObjects
    self.testFile = HTMLFile()
    self.SynchronizationChecks()
    self.testFile.Close()

  def SynchronizationChecks(self):
```

```
project = self.projectObjects.Get(GC.ART_UNE_PROJECT)[0]
metrics = project.metricsValues.keys()
# Check 1.1 ************************************************
title = 'Every actor in the context model for the current system '\
    'is a candidate actor for the context model of the future system.'
self.testFile.AddCheck('Check1.1', title)
list1 = project.metricsValues['Check_1_1_Part1'][0]
remove = 'Do you want to remove these actors to create your new system?'
list2 = project.metricsValues['Check_1_1_Part2'][0]
add = 'Do you want to add these actors to create your new system?'
if (list1 != None) and (list2 != None):
    self.TextForm(remove, add, list1, list2)
else:
    self.testFile.AddSpecial('At least one of the context model'\
                    'is missing in the project.', 1)
    self.testFile.CloseTable()
# Check 1.2 ************************************************
title = 'Every Data flow in the context model for the current system '\
        'is a candidate data flow for the context model of the future system.'
self.testFile.AddCheck('Check1.2', title)
list1 = project.metricsValues['Check_1_2_Part1'][0]
remove = 'Do you want to remove these data flows to create your new system?'
list2 = project.metricsValues['Check_1_2_Part2'][0]
add = 'Do you want to add these data flows to create your new system?'
if (list1 != None) and (list2 != None):
    self.TextForm2(remove, add, list1, list2)
else:
    self.testFile.AddSpecial('At least one of the context model'\
                    'is missing in the project.', 1)
    self.testFile.CloseTable()
# Check 1.3 ************************************************
title = 'Every use case in the use case diagram for the current'\
        'system is a candidate for a use case in the use case diagram '\
        'for the future system.'
self.testFile.AddCheck('Check1.3', title)
list1 = project.metricsValues['Check_1_3_Part1'][0]
remove = 'Do you want to remove these use cases to create your new system?'
list2 = project.metricsValues['Check_1_3_Part2'][0]
add = 'Do you want to add these use cases to create your new system?'
if (list1 != None) and (list2 != None):
    self.TextForm(remove, add, list1, list2)
else:
    self.testFile.AddSpecial('At least one of the use case diagram '\
                    'is missing in the project.', 1)
```

```
    self.testFile.CloseTable()
# Check 1.4 ************************************************
title = 'Every adjacent actor which communicates directly '\
        'with the technical system (level 1) in the context model '\
        'for the current system is a candidate actor for the use '\
        'case diagram for the current system.'
self.testFile.AddCheck('Check1.4', title)
list1 = project.metricsValues['Check_1_4_Part1'][0]
remove = 'These adjacent actors from the context model of the current '\
        'system are not present in the Use Case diagram of the current system:'
list2 = project.metricsValues['Check_1_4_Part2'][0]
add = 'These actors from the Use Case diagram of the current system'\
        ' are not present in the Context model of the current system:'
if (list1 != None) and (list2 != None):
    self.TextForm(remove, add, list1, list2)
else:
    self.testFile.AddSpecial('The context model or the use case diagram'\
                        'is missing in the project.', 1)
    self.testFile.CloseTable()
# Check 1.5 ************************************************
self.testFile.AddCheck('Check 1.5', 'The system boundary in the use case '\
        'model for the current system model should \nbe the same as the '\
        'boundary between levels 1 & 2 in the context model for the '\
        'current system.')
self.testFile.CloseTable()
# Check 1.6 ************************************************
self.testFile.AddCheck('Check 1.6', 'For every data flow from or to level 1 '\
        '(or one of the sub systems within level 1) of the context model '\
        'of the current system, there should be a corresponding line in '\
        'the use case diagram for the current system indicating '\
        'involvement of the relevant actor in the relevant use case.')
liste = project.metricsValues['CurrentUseCaseInfo'][0]
curAdjDataFlow = project.metricsValues['CurrentAdjDataFlowsInfo'][0]
level234 = project.metricsValues['CurrentLevel234'][0]
    if (liste != None) and (curAdjDataFlow != []):
        actorDict =
        for actor in liste:
            actor = actor.split($')
            actorDict[actor[0]] = actor[1:]
        i = 0
        line = 1
        self.testFile.AddLine('These data flows of the current context model '\
                'do not have corresponding use cases \n'\
                'in the use case diagram:', line)
```

```
bidirectional = 0
while i < (len(curAdjDataFlow)):
    contextInformations = curAdjDataFlow[i].split('')
    senders = contextInformations[1].split($')
    receivers = contextInformations[2].split($')
    actor = ''
    if senders[1] != '':
        # bidirectional data flow
        bidirectional = 1
        if senders[0] in level234:
            actor = senders[0]
        elif senders[1] in level234:
            actor = senders[1]
    else:
        # unidirectional data flow
        if senders[0] in level234:
            actor = senders[0]
        elif receivers[0] in level234:
            actor = receivers[0]
    if actorDict.has_key(actor):
        # search for corresponding use case in those of this actor
        contextInformations = contextInformations[0].split('/')
        for info in contextInformations:
            res = 0
            for uc in actorDict[actor]:
                res2 = self.StringCompared(info.upper(), uc.upper())
                if res2 > 70:
                    res = res+1
            if ((bidirectional==0) and(res<1)):
                self.testFile.AddCell( info+'\nsent by: '+
                    str(senders[0:-1])+'\nreceived by: ' +
                    str(receivers[0:-1]), line)
                line = line+1
            elif ((bidirectional==1)and(res<2)):
                if res == 0:
                    self.testFile.AddCell( info+'\nsent by: '+
                        str(senders[0:])+'\nreceived by: ' +
                        str(receivers[0:])+'\n' +
                        (2 use cases missing (1 for each direction))', line)
                elif res ==1:
                    self.testFile.AddCell( info+'\nsent by: '+
                        str(senders[0:])+'\nreceived by: ' +
                        str(receivers[0:])+'\n' +
                        (1 use case missing (for 1 of the directions))', line)
```

```
                    line = line+1
            else:
                self.testFile.AddCell(contextInformations[0]+'\nsent by: '+
                    str(senders[0:-1])+'\nreceived by: ' + str(receivers[0:-1]) +
                    '\n(cause actor \"'+actor+
                    '\"not found in Use Case diagram)', line)
                line = line+1
            bidirectional = 0
            i = i+1
    else:
        self.testFile.AddSpecial('Context model or use case diagram'\
            ' missing in the project.', 1)
    self.testFile.CloseTable()
    # Check 1.7 ************************************************
    self.testFile.AddCheck('Check 1.7', 'Every adjacent actor (level 2, 3 or 4) '\
            'which communicates directly with the technical system (level 1) '\
            'in the context model for the future system is a candidate actor for '\
            'the use case diagram for the future system.')
    list1 = project.metricsValues['Check_1_7_Part1'][0]
    remove = 'These adjacent actors from the context model of the future '\
            'system are not present in the Use Case diagram of the future system:'
    list2 = project.metricsValues['Check_1_7_Part2'][0]
    add = 'These actors from the Use Case diagram of the future system'\
            ' are not present in the Context model of the future system:'
    if (list1 != None) and (list2 != None):
        self.TextForm(remove, add, list1, list2)
    else:
        self.testFile.AddSpecial('The context model or the use case diagram '\
            'is missing in the project.', 1)
        self.testFile.CloseTable()
    # Check 1.8 ************************************************
    self.testFile.AddCheck('Check 1.8', 'The system boundary in the use case '\
            'model for the future system model should be the same as the '\
            'boundary between levels 1 & 2 in the context model for the '\
            'future system.')
    self.testFile.CloseTable()
    # Check 1.9 ************************************************
    self.testFile.AddCheck('Check 1.9', 'For every data flow from or to level 1 '\
            '(or one of the sub systems within level 1) of the context model of the '\
            'future system, there should be a corresponding line in the use case '\
            'diagram for the future system indicating involvement of the '\
            'relevant actor in the relevant use case.')
    liste = project.metricsValues['FutureUseCaseInfo'][0]
    futAdjDataFlow = project.metricsValues['FutureAdjDataFlowsInfo'][0]
```

```
level234 = project.metricsValues['FutureLevel234'][0]
if (liste != None) and (futAdjDataFlow != []):
    actorDict =
    for actor in liste:
        actor = actor.split($')
        actorDict[actor[0]] = actor[1:]
    line = 1
    self.testFile.AddLine('These data flows of the future context model '\
        'do not have corresponding use cases in the use case diagram:', line)
    i = 0
    line = line + 1
    while i < (len(futAdjDataFlow)):
        contextInformations = futAdjDataFlow[i].split('')
        senders = contextInformations[1].split($')
        receivers = contextInformations[2].split($')
        actor = ''
        bidirectional=0
        if senders[1] != '':
            # bidirectional data flow
            bidirectional = 1
            if senders[0] in level234:
                actor = senders[0]
            elif senders[1] in level234:
                actor = senders[1]
        else:
            # unidirectional data flow
            if senders[0] in level234:
                actor = senders[0]
            elif receivers[0] in level234:
                actor = receivers[0]
        if actorDict.has_key(actor):
            # search for corresponding use case in those of this actor
            contextInformations = str(contextInformations[0]).split('/')
            for info in contextInformations:
                res = 0
                for uc in actorDict[actor]:
                    res2 = self.StringCompared(info.upper(), uc.upper())
                    if res2 > 70:
                        res = res + 1
            if (bidirectional == 0)and(res!=1):
                self.testFile.AddCell( info+'\nsent by: '+str(senders[0:])+
                        '\nreceived by: ' + str(receivers[0:]), line)
                line = line+1
            elif ((bidirectional==1)and(res<2)):
```

```
                if res == 0:
                    self.testFile.AddCell( info+'\nsent by: '+str(senders[0:])+
                        '\nreceived by: ' + str(receivers[0:])+
                        '\n(2 use cases missing (1 for each direction))', line)
                elif res ==1:
                    self.testFile.AddCell( info+'\nsent by: '+str(senders[0:])+
                        '\nreceived by: ' + str(receivers[0:])+
                        '\n(1 use case missing (for 1 of the directions))', line)
                line = line+1
            else:
                self.testFile.AddCell(contextInformations[0]+'\nsent by: '+
                    str(senders[0:])+'\nreceived by: ' + str(receivers[0:]) +
                    '\n(cause actor "+actor+'\"not found in Use Case diagram)', line)
                line = line+1
            bidirectional = 0
            i = i+1
    else:
        self.testFile.AddSpecial('Contextmodel or use case diagram '\
            missing in the database.', 1)
    self.testFile.CloseTable()
    # Check 1.10 ************************************************
    title = 'Services and functions related to use cases in the use case diagram '\
            'for the future system should map to system level requirements, i.e. '\
            'high level functional and non functional requirements in the '\
            'requirements database.'
    self.testFile.AddCheck('Check 1.10', title)
    self.testFile.CloseTable()
    # Check 2.1 *************************************************
    title = 'All external actors in the i* SD model of the current system should '\
            'correspond to actors in the use case descriptions for the current system.'
    self.testFile.AddCheck('Check 2.1', title)
    line = 1
    liste = project.metricsValues['Check_2_1'][0]
    if (liste != None) :
        self.testFile.AddLine('These external actors in the Strategic dependency '\
            'model for the current system are not present in the use case'\
            'descritpions:', line)
        line = line + 1
        i=0
        while i < len(liste):
            self.testFile.AddCell(liste[i], line+i+1)
            i = i+1
    else:
```

```
    self.testFile.AddSpecial('SD model or use case descriptions '\
        'missing in the project.', 1)
self.testFile.CloseTable()
# Check 2.2 ************************************************
title = 'All external actors in the i* SD model of the current system '\
        'are candidate actors for the i* SD model of the future system.'
self.testFile.AddCheck('Check 2.2', title)
remove = 'Do you want to remove these external actors from the SD '\
        'model to create the new system?'
list1 = project.metricsValues['Check_2_2_Part1'][0]
add = 'Do you want to add these actors to create the new system?'
list2 = project.metricsValues['Check_2_2_Part2'][0]
if (list1 != None) and (list2 != None):
    self.TextForm(remove, add, list1, list2)
else:
    self.testFile.AddSpecial('At least one of the SD models is missing.', line)
self.testFile.CloseTable()
# Check 2.3 ************************************************
title = 'All dependencies in the i* SD model of the current system are '\
        'candidate dependencies for the i* SD model of the future system.'
self.testFile.AddCheck('Check 2.3', title)
line = 1
list1 = project.metricsValues['Check_2_3_Part1'][0]
list2 = project.metricsValues['Check_2_3_Part2'][0]
if (list1 != None) and (list2 != None):
    self.testFile.AddLine('Do you want to remove these dependencies '\
        'to create the new system?', line)
    i = 0
    while i < len(list1):
        info = list1[i].split($')
        type = info[0]
        dep = info[1]
        sender = info[2]
        receivers = info[3]
        self.testFile.AddCell(type + ' dependency \" + dep+
                '\'\n- depender:\n ' +sender +
                '\n- dependees:\n ' + receivers, line)
        line = line + 1
        i = i+1
    line = line +1
    self.testFile.AddLine('Do you want to add these dependencies '\
        'to create the new system?', line)
    i=0
    while i < len(list2):
```

```
            info = list2[i].split($')
            type = info[0]
            dep = info[1]
            sender = info[2]
            receivers = info[3]
            self.testFile.AddCell(type + ' dependency \" + dep
+'\'\n- depender:\n ' +
                    sender + '\n- dependees:\n ' + receivers, line)
            line = line + 1
            i = i+1
    else:
        self.testFile.AddSpecial('At least one of the SD model is missing.', line)
    self.testFile.CloseTable()
    # Check 2.4 ************************************************
    title = 'All actions in use case descriptions for the current system are '\
            'candidate actions for use case descriptions of the future system.'
    self.testFile.AddCheck('Check 2.4', title)
    line = 1
    list1 = project.metricsValues['CurrentUseCaseActions'][0]
    remove = 'Do you want to remove these actions to create the new system?'
    list2 = project.metricsValues['FutureUseCaseActions'][0]
    add = 'Do you want to add these actions to create the new system?'
    if (list1 != None) and (list2 != None):
        list11=[]
        for a in list1:
            res = 0
            for a2 in list2:
                r = self.StringCompared(a.upper(), a2.upper())
                if r>res:
                    res = r
            if res>65:
                pass
            else:
                list11.append(a)
        list22=[]
        for a in list2:
            res = 0
            for a2 in list1:
                r = self.StringCompared(a.upper(), a2.upper())
                if r>res:
                    res=r
            if res>65:
                pass
            else:
```

```
                list22.append(a)
        self.TextForm(remove, add, list11, list22)
    else:
        self.testFile.AddSpecial('At least one of the model doesn\'t have use'\
            ' case descriptions including actions in the database.', 1)
        self.testFile.CloseTable()
    # Check 2.5 ************************************************
    title = 'All variations in use case descriptions for the current system are '\
            'candidate variations or use cases for the future system.'
    self.testFile.AddCheck('Check 2.5', title)
    line = 1
    list1 = project.metricsValues['CurrentUseCaseVariations'][0]
    remove = 'Do you want to remove these variations to create the '\
            'new system?'
    list2 = project.metricsValues['FutureUseCaseVariations'][0]
    add = 'Do you want to add these variations or use cases to create'\
            ' the new system?'
    if (list1 != None) and (list2 != None):
        list11=[]
        for a in list1:
            res = 0
            for a2 in list2:
                r = self.StringCompared(a.upper(), a2.upper())
                if r>res:
                    res = r
            if res>65:
                pass
            else:
                list11.append(a)
        list22=[]
        for a in list2:
            res = 0
            for a2 in list1:
                r = self.StringCompared(a.upper(), a2.upper())
                if r>res:
                    res=r
            if res>65:
                pass
            else:
                list22.append(a)
        self.TextForm(remove, add, list11, list22)
    else:
        self.testFile.AddSpecial('At least one of the model doesn\'t have use'\
            ' case descriptions including variations in the database.', 1)
```

```
    self.testFile.CloseTable()
# Check 2.6 ************************************************
title = 'All external actors in the i* SD model of the future system should '\
        'correspond to actors in the use case descriptions for the future system.'
self.testFile.AddCheck('Check 2.6', title)
ucDescrActors = project.metricsValues['FutureUseCaseDescrActors'][0]
if ucDescrActors == []:
    self.testFile.AddSpecial('All actors fields are empty in the use case '\
        'descriptions', 1)
liste = project.metricsValues['Check_2_6'][0]
line = 2
if (liste != None) :
    self.testFile.AddLine('These external actors in the Strategic dependency '\
        'model for the future system are not present in the use case '\
        'descriptions:', line)
    line = line + 1
    i=0
    while i < len(liste):
        self.testFile.AddCell(liste[i], line+i+1)
        i = i+1
else:
    self.testFile.AddSpecial('SD model or use case descriptions missing '\
        'in the project.', 1)
self.testFile.CloseTable()
# Check 2.7 ************************************************
title = 'For all task dependencies identified in the i* SD model of the '\
        'future system, which represent tasks carried out by actors in the '\
        'use case diagram, there should be a part of a use case description '\
        'which describes how those tasks are carried out.'
self.testFile.AddCheck('Check 2.7', title)
line = 1
list1 = project.metricsValues['FutTaskDependencies'][0]
if list1 != None:
    self.testFile.AddLine('Here is a list of the task dependencies in '\
        'the SDModel for the future system:', line)
    i = 0
    while i<len(list1):
        info = list1[i].split($')
        dep = info[1]
        sender = info[2]
        receivers = info[3]
        self.testFile.AddCell(dep +'\'\n- depender:\n ' +
                sender + '\n- dependees:\n ' + receivers, line)
        line = line + 1
```

```
            i = i+1
    else:
        self.testFile.AddSpecial('SD model or use case descriptions missing.', line)
    self.testFile.CloseTable()
    # Check 2.8 ************************************************
    title = 'All requirements associated with use cases using the RESCUE '\
            'use case template should be stored in the requirements database.'
    self.testFile.AddCheck('Check 2.8', title)
    line = 1
    self.testFile.AddLine('These requirements are not described in the'\
            'requirements database:', line)
    self.testFile.CloseTable()

  def TextForm(self, remove, add, list1, list2):
    line = 1
    self.testFile.AddLine(remove, line)
    i=0
    while i < len(list1):
        self.testFile.AddCell(list1[i], line+i+1)
        i = i+1
    line = i
    self.testFile.AddLine(add, line)
    i = 0
    while i < len(list2):
        self.testFile.AddCell(list2[i], line+i+1)
        i = i+1
    self.testFile.CloseTable()

  def TextForm2(self, remove, add, list1, list2):
    line =1
    self.testFile.AddLine(remove, line)
    i=0
    while i< len(list1):
        informations = list1[i].split('')
        self.testFile.AddCell(informations[0]+'\nsent by: '+
            str(informations[1].split($')[0:-1])+'\nreceived by: '+
            str(informations[2].split($')[0:-1]), line+i+1)
        i = i+1
    line = i
    self.testFile.AddLine(add, line)
    i=0
    while i< len(list2):
        informations = list2[i].split('')
        self.testFile.AddCell(informations[0]+'\nsent by: '+
```

```
            str(informations[1].split($')[0:-1])+'\nreceived by: '+
            str(informations[2].split($')[0:-1]), line+i+1)
        i = i+1
    self.testFile.CloseTable()

def StringCompared(self, string1, string2):
    "'compare if two given strings are equal; case sensitive"'
    words1 = string1.split(' ')
    for w in words1:
        w=self.RemoveSpace(w)
    words2 = string2.split(' ')
    for w in words2:
        self.RemoveSpace(w)
    i = 0
    res = 0
    for w in words1:
        for w2 in words2:
            if w == w2:
                res = res+1
                break
    if words1[len(words1)-1]==":
        res = ((float(res)/(len(words1)-1))*100)
    else:
        res = ((float(res)/len(words1))*100)
    return res

def RemoveSpace(self, string):
    i = string.find(' ')
    if i!=-1:
        if i==len(string):
            string=string[:i]
        else:
            string=string[:i]+string[i+1:]
        string=RemoveSpace(string)
    else:
        return string
```